UNIVERSITY OF OSLO
Department of Informatics

Automatic scaling of Cassandra clusters

Master thesis

Tor Andreas
Baakind

April 30, 2013

Automatic scaling of Cassandra clusters

Tor Andreas Baakind

April 30, 2013

Abstract

The purpose of this thesis is to create an automatic scaling implementation for Cassandra clusters. The automatic scaler should never lower the overall performance of the cluster in a way that results in a bad user experience. It should also be able to successfully scale up and down nodes, and the cluster should continue as if nothing happened. Last but not least, it is desirable that the automatic scaler performs equally, or better than, the person who is in charge of administrating the database.

In this thesis we have developed an early version of an autoscaler that may run alongside a Cassandra instance. The implementation is split into two separate implementations: a master-, and an agent-implementation. The master will be deployed to the same server as the application using the cluster, even though this is not required. The agent implementation will be deployed to, and run alongside, all nodes that are a part of the cluster. The agent will monitor the node's resource usage, and send messages back to the master if the usage increases above, or decreases below certain thresholds.

We performed a set of test cases to prove that the implementation works as intended. The test cases recorded the nodes resource-usage to determine the impact our implementation makes to the overall performance.

Acknowledgments

I would like to thank my supervisors, Ketil Velle and Dag Langmyhr, for their guidance and valuable feedback. This thesis would not have been completed without them. I would also like to thank Jørgen Sørensen for proofreading the thesis.

I also want to thank my girlfriend Anniken, for being so supportive and understanding throughout the thesis work. And finally, I would like to thank my parents for believing in me.

Contents

List of Tables

List of Figures

Part I

Introduction

Chapter 1

Introduction

1.1 Problem definition

As of today it does not exist any implementation able to automatically scale a Cassandra cluster base on each node's resources. Netflix has created an implementation named *Priam* that among other things are able to efficiently double the size of the Cassandra cluster by pairing each new node with an already existing node and share its load[29].

Although Netflix probably has a very efficient implementation for doubling the size of a cluster, our implementation tries to solve the problem by inserting each new node at *hotspots* that occur. A hotspot is a location of the cluster that has a higher density of data than the rest. When a new node is inserted into an already existing token range, the token range is shared equally among the new- and the already existing node. Even though they shared the token range in-between themselves, the inserted data will not be distributed equally. It will not be equally distributed since the data is stored based on the *hashed value* of the key. The key is the unique identifier used to identify the data set. Since the outcome of the hashing algorithm is fairly random, it is hard to prepare the cluster in order for it not being unbalanced.

Inserting nodes continuously at hotspots that occur will result in an unbalanced cluster after some time. Hopefully the cluster will not end up being too unbalanced, and decrease the overall performance. Unlike Priam, the implementation will be able to scale down when the cluster operates on too many nodes. The cluster operates on too many nodes when nodes may be removed without affecting the performance of the cluster or the applications using it.

Scaling of a Cassandra cluster would most likely require a lot of resources and bandwidth to transfer data from one node to another. This will lower the overall performance of the cluster, and weakens the main focus of Cassandra: *extreme performance and scalability*. Therefore the scaling should be triggered at low-peak hours, since the nodes will be able to handle the scaling combined with the incoming load from external sources.

1.2 Contribution

It would ease the work for the database administrator if the scaling process where automated. Today the database administrator has to carefully monitor the nodes current health, and detect deviations from the normal behavior. Whenever a deviation is detected, the database administrator have to take care of it, and eventually scale up or down nodes depending on his or hers decision. There exist tools for easier monitoring of the cluster, e.g., The OpsCenter from DataStax[13]. The OpsCenter visualize the cluster and includes information about each node's current status, and in-depth monitoring of each nodes performance and load.

By automating the scaling process, the database administrator would not have to consider if there is necessary to scale up or down nodes in the cluster. To understand if the cluster should be scaled up or down may sometimes require domain knowledge. Sometimes there may be applications that have periods where the read- and write-requests are very intense, compared to what is usual. In these cases, the database administrator will need to have domain knowledge in order to scale up enough nodes ahead of the event, and eventually know when to scale down. By automating this process, a scale-up may be triggered within seconds, and a scale down triggers when the event is over. This allows such events to happen anytime since the automatic scaler will detect the increase or decrease in traffic, and act accordingly.

For the automatic scaling implementation to be successful, the criteria would of course be that the implementation performs equally- or better than the average database administrator is able to perform. It should also be able to respond quicker, and hopefully provide better performance than a database administrator will be able to do.

For the community to fully take advantage of the implementation, it has to be developed even further. Currently it is a very simple and straightforward implementation that only looks for high/low memory- and disk usage, and trigger the scaling process based on the provided thresholds. It currently does not consider if it scales up or down during peak-hours, which may lead to a sudden drop in performance during critical periods e.g., releases or sales.

1.3 Outline

The thesis is organized as follows. Part II contains the background material, and the motivation for the project. It describes the technology that is necessary to understand in order to develop the implementation. Part II also contain a brief introduction to the work related to the project. Part III describes the project, and the work that has been done. It describes the goals for the thesis work, and the methodology used while working with the thesis. Part III also describes the design, and implementation work in detail. The test cases that have been performed are also described, and the results that were recorded are visualized through graphs with description.

Part IV assesses the design, and implementation, of the automatic scaler developed during the thesis work. It verifies the final result to see if it satisfies the goals of this thesis. The part also contains an analysis of the test results, to determine the severity of the impact made to overall performance of the Cassandra cluster. Finally, it contains a summary of the thesis, and a list of future work that should be done before the implementation is deployed to a real cluster.

Part II

Background

Chapter 2

Motivation

Together with the new era of Internet companies like Google, Amazon and Facebook, came problems and difficulties considering the database management. They all struggled with one main problem: The huge amount of data passing through the Internet at a daily basis, which increases every day. The traditional *RDBMS* (Relational Database Management Systems) does not manage to store all the data and provide good performance the way they are designed[26]. RDBMS were originally built to work on a single machine, not act as a cluster of servers like the companies needed for parallelism and fast real-time response

The most important factor for Google, Amazon and Facebook today would probably be to have their services online at all times, so their customers never experience any down time. Today almost every internet user around the world expect any service to be available at any time, and also respond to any request within a fair amount of milliseconds. Therefore these companies always have to focus on their performance and response time to keep up with the increasing amount of data and the current (and future) requirements from their customers. If e.g., a service delivered by Amazon experience poor performance, and maybe goes offline, it may results in a lot of customers leaving, as it is extremely easy to turn away for another service on the internet. Therefore Google, Facebook, Amazon and all other companies which delivers real-time services to a large amount of users cannot afford to sit back and envy the number of users currently paying for their services, but have to always be up front, handling problems and always trying to be better.

Since good performance and fast response is what makes up these companies, and with the incredible large amount of data which they have to handle every single day, they had to think of new ways to store and structure the data. The traditional relational database systems were good at structure smaller amounts of data for e.g., a banking institution. But when it comes to large and unstructured amounts of data, the RDMS is not the right choice. The RDMS has to pre-declare schemas that tell which data to store, how to store it, and what kind of attributes that exists for the data.

Relational database systems also provide ACID-compliant transactions, which means that what is written is what is retrieved by the next

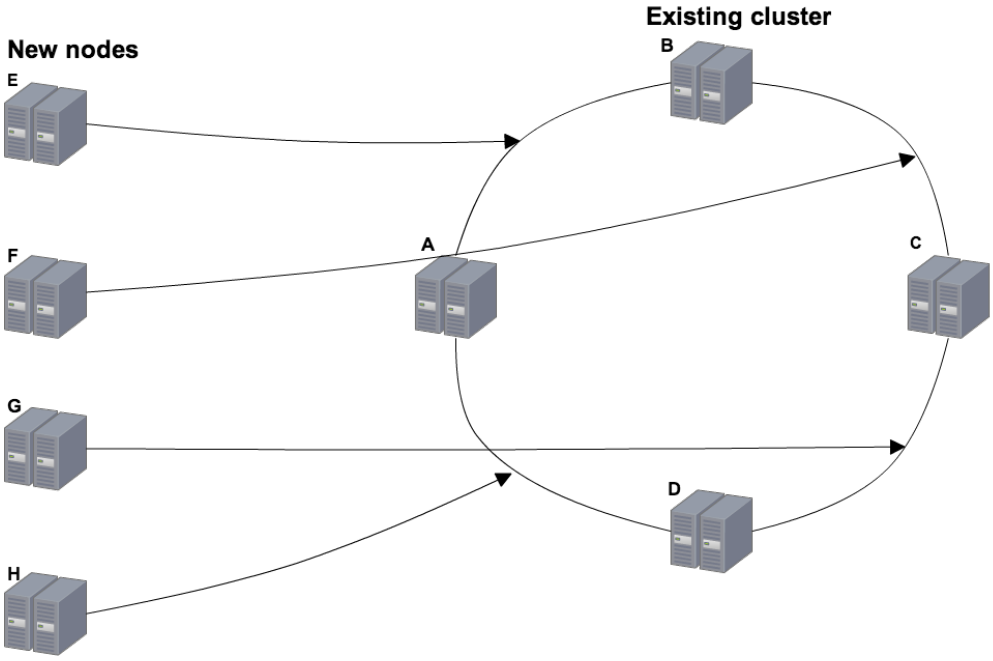


Figure 2.1: Double the cluster-size is recommended when scaling up a Cassandra cluster

transaction. It locks the data that is currently being manipulated in order to prevent other transactions from making changes to it. This leads to a lot of overhead, and may cause write-intensive systems to almost freeze if multiple users are updating the same data at the same time in the database.

As a result of this, these internet-companies came up with new database models solving their problems. These database models are often referred to as *NoSQL*. NoSQL is a wide acronym for the non-relational databases, which cannot be labeled as a relational database.

One of the most known NoSQL databases today is Cassandra. Cassandra is a one of the best databases today when it comes to scalability and high availability without lowering performance [30]. Cassandra supports ad-hoc scaling, which means that any new nodes may connect to the cluster by interacting with an already existing node. As Cassandra allows dynamically scaling of clusters, it opens up the possibility to deliver "infinite" capacity by increasing with n nodes whenever the cluster is about to run out of space. To prevent the cluster from being unbalanced, it is recommended to always extend with $numberOfNodesx2$, where $numberOfNodes$ is the number of nodes in the cluster. E.g., if there are 4 active nodes in the cluster and the cluster scales up, it should scale up to 8 nodes. To prevent too much data moving across multiple nodes, and to keep the scaling as isolated as possible to the nodes involved, each new node gets paired with one of the existing nodes, so that each existing node pairs with 1, and only 1 node as seen in figure 2.1. By default, node E pairs with node B, node F pairs with node C, node G pairs with node D, and node H pairs with node A. When the node pairs up, they share the already existing token-range

of the existing node, and each node ends up with 50% of the token-range. Cassandra's default token-assigner cut the responsible token-range in half, e.g., if the existing node is responsible for #0000 - #3999, the node will continue to serve the token-range #2000 - #3999, while the new node will be responsible for the token range #0000 - #1999 or vice versa.

It is a preferred feature to support ad-hoc scaling while the cluster being online, since there is never a good idea to bring down an active cluster to increase or decrease the available space and computing power. Even though Cassandra supports real-time ad-hoc of new nodes, the operation has to be performed manually by the database administrator. There is no solution that automatically keeps track of the cluster's health and initiate scaling based on disk-, memory- or CPU-usage over time. Our implementation is meant to fulfill this need. It monitors each individual nodes health, and determines if the cluster needs to be scaled up or down based on the monitored values. It will not initiate scaling directly after a threshold is breached, as this will result in a lot of up- and downscaling which will lower performance and increase network traffic more than necessary. It will monitor the breaches over time, and if the breach occurs continuously for a given time, the implementation will either scale up- or down nodes depending on the breach-type.

Chapter 3

NoSQL

NoSQL is a term that is used to describe database systems which is different from the traditional relational database systems. The NoSQL term has not been officially defined yet, although some people think that it means *Not only SQL*. Eben Hewitt has his own explanation of what NoSQL is all about in his book *Cassandra: The Definite Guide*[20]. Hewitt think that the NoSQL term should not be used to cover all databases that are not relational, as they do not share the same implementation, goals, features, advantages and/or disadvantages. Each database system was designed for different purposes and has different focus. Hewitt believes that - "comparing NoSQL to relational is really a shell game". What he probably means is that NoSQL cannot be directly compared to a relational database as it covers many different variants of non-relational databases. Each non-relational database that is covered by the NoSQL term should be compared against the relational database for the comparison to be correct.

Since the end of the 21st century there have been designed many different database solutions that are categorized as *NoSQL solutions*. The majority of the databases are inspired by either Google's BigTable[10] which is Google's own database model used for many of their services, or Amazon Dynamo, which is Amazon's database model used for handling their shopping cart functionality[14]. Some solutions are directly related to one of these, some are a mix of both, while others are something quite different. To narrow down the jungle of different NoSQL solutions currently existing they may be grouped into categories according to their data model. The three most essential categories that will be explained in this thesis are: *Extensible record stores*, *Key-value stores* and *Document stores*[9]. This thesis will only cover a brief summary of each of the categories as well as graph databases, since the main focus of this thesis is Cassandra.

3.1 Database transactions

We have mentioned that large companies like Facebook, Google and Amazon were in need of a database model that could handle large amount of data spread across thousands of servers all over the world, queried

million times a day. The amount of data they process and index every day is rapidly increasing, and the traditional relational database model systems (RDBMS) are not built for tasks like search engines, social networks etc.

Performance is one of the most important features when dealing with the amount of data e.g., Google indexes just for their search engine. Since relational databases would perform badly if set to solve tasks like indexing documents for a global search engine, they were forced to think new and create solutions that were able to handle large amount of data each day as well as the increasing amount of users which were, and still are, using their services. A criterion they had while designing their new solutions may have been that the response time of the system could not be lowered due to the amount of data or the change of database system; it had to be equal, or even better than before.

The biggest bottleneck of a relational database system is their Acid-Compliant transactions, which guarantees that the data written to the database is what is retrieved by the next transaction.

3.1.1 The ACID sacrifice

ACID is an acronym for *Atomicity*, *Consistency*, *Isolation* and *Durability* [20], and is one of the key features of a relational database system. The ACID guarantee ensures that the data written to the database, is the same data retrieved by the next transaction. To achieve ACID-Compliant transactions, data has to be locked, and only one transaction may be able to manipulate each data set at a time. Since a subset of the data is locked due to manipulation, other transactions will have to wait until the locks are released. As transactions are queued, it may occasionally lead to bottlenecks. The database system will use a lot of resources just to apply and hold the locks as well, which means that there will be less resources for everything else, and the overall performance will be lowered. The four transaction rules that are a part of the ACID guarantee are:

- ***A**tomicity* - If there is a set of operations to be performed at once, either all of them occurs, or none. E.g., you only want to update your database if all user details are stored in the respective tables for address, contact information, login-details etc. All the insertions, and possible updates are executed within the same transaction and if one of them fails, the database is rolled back to its previous state and nothing is stored or updated.

- ***C**onsistency* - Make sure the data written to the database follows a set of pre-defined rules like constraints, data types etc. If the transaction(s) supposed to be performed were successful, the database system moves the whole database into a new state with the new and updated data.

- ***I**solation* - Isolate and lock parts of the database that is manipulated or in use by the transaction(s). By isolating the data involved, the transaction manager makes sure that no other transactions updates,

or retrieves, the data while it is being updated. By isolating the data involved, the database system makes sure that at the point of updating, no one else than the current transaction is able to read or update the data involved. When the update is successful, the locks are released and other transactions may read or update the data. This guarantees that what is stored in the database is what is retrieved. The complexity of the isolation increases when the system is distributed, as it will require a lot more resources and coordinating to perform locking across multiple servers.

One way to increase performance while updating data is to make snapshots of the data currently being manipulated. If there are any other transactions trying to access the data being updated, they may read, but not update, the snapshot instead of waiting for the transaction manipulating the data to finish.

- ***D**urability* - Keeps track of committed transactions to the database. If a transaction is committed to the database, it should not be lost if e.g., the power is cut. The transaction manager performs regular backups of the data and the transaction logs in case of something unexpected happens. The transaction logs are used to rollback data if something went wrong by reversing the operations done, but also continue from the last successful operation if e.g., the power were cut.

The majority of the NoSQL solutions that exists today have sacrificed the ACID guarantee in order to achieve sufficient response time. In most cases, the ACID guarantee may not even be required. It may not be important if your friends wall post on Facebook shows up in your feed a second after it was posted, or if your twitter-post does not reach all your friends at the exact same moment as it was posted. What matters are that no one has to sit and wait for the news feed to load because the database system performs badly. The ACID guarantee was sacrificed in order to achieve the appropriate performance and response time, as it consumes a lot of the overall processing time.

However there are situations where the ACID guarantee and an Acid-Compliant database system is required. E.g., a financial institution executes a bank transaction. As a bank transaction transfer peoples money, it is extremely important that the data is consistent and nothing goes wrong. Bank transactions have no rooms for misleading data or inconsistent data (although companies like Visa have special cases which let users spend money, even if their account is not accessible at the time of the transaction, but that is outside the scope of this thesis).

Since most NoSQL solutions sacrifices the ACID guarantee in order to achieve better performance, they are following another set of rules. A computer scientists called Eric Brewer [7] came up with a conjecture in 2000, which two years later were proven and established as a theorem, *Brewer's CAP theorem*.

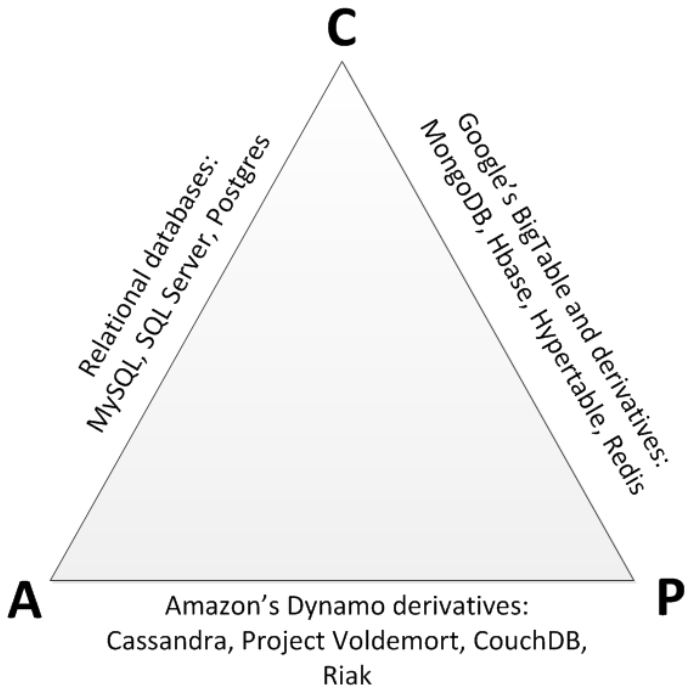


Figure 3.1: Positioning of different databases according to the Brewer's CAP theorem.

3.2 Brewer's CAP theorem

Brewer's CAP theorem is a theorem that groups different database systems based on their abilities. Since the majority of the NoSQL solutions were forced to sacrifice the ACID guarantee to focus on more important features for their particular needs, the CAP theorem were invented to group the different NoSQL solutions together. There are a lot of different NoSQL solutions that have been created the last couple of years that serves quite different needs. Everything from social network feeds like the Facebook news feed, to people's relationships to each other, companies etc. like connections at LinkedIn. *CAP* is an acronym that stands for[5]:

- ***C**onsistency* - All nodes connected see the same data at the same time.
- ***A**vailability* - If a request is sent to the database, it is guaranteed that a response is received, even if the request were not successful.
- ***P**artition tolerance* - The system has no single point of failure. If one node fails, the data is still accessible from another node, and the system will run as if nothing happened.

Hewitt states in his book about Cassandra that "Brewer's theorem is that in any given system, you can strongly support only two of the three"[20]. The statement concludes that a system cannot strongly support Consistency, Availability and Partition tolerance at the same time. A system

cannot be 100% consistent and available at any given time if it is distributed across multiple nodes. If new data is inserted or existing data is updated at one node, due to physical barriers, there will take a few milliseconds or seconds to make the newly retrieved data available at the other nodes as well. That is why the system is called *eventual consistent*. Hewitt explains three different levels of consistency in his book about Cassandra[21]:

- ***Strict consistency*** - (sometimes called sequential consistency) Requires all data returned from the database to be the most up-to-date data available. To achieve strict consistency across multiple nodes throughout multiple data centers around the world, a global timer mechanism will be required to put a timestamp on the data and operations applied to the system. Strict consistency may be used by e.g., financial institutions or e-commerce websites as their data have to be consistent at all times. The main advantage achieved from strict consistency is the guarantee that the data returned will always be valid. On the other side, the main disadvantage is the sacrifice of performance because the system will have to check multiple nodes for the most up-to-date data.

- ***Casual consistency*** - As Hewitt states, casual consistency is a slightly weaker condition than strict consistency. To increase performance, this level of consistency gets rid of the global synchronize clock and timestamp checking that generates a lot of overhead for systems using strict consistency.

- ***Eventual (weak) consistency*** - All nodes within the system will eventually have the most up-to-date data and be consistent, but there is no guarantee for when it happens. Eventual consistency may be necessary for websites or services that requires fast response and the most up-to-date data is not necessarily required - e.g., Facebook's wall or Google's search.

Figure 3.1 on the preceding page is a remake of Hewitt's figure from his book. The figure visualizes which parts of the Brewer's CAP theorem the most known NoSQL solutions support with the out-of-the- box configuration. As mentioned earlier, it is not possible to support all three features of the CAP theorem strongly. It is only possible to support two out of three, while its possible to partially support the third. E.g., Cassandra supports *Availability* and *Partition tolerance*. However, Cassandra also supports *eventual consistency* where data is consistency within a reasonable amount of time.

Some database solutions support the **A**vailability and **P**artition tolerance of the Brewer's CAP theorem. These database solutions do not support consistency the same way the relational database systems does, but they may support eventual consistency were data will be replicated to the remaining nodes at any given time, as Cassandra does. These systems, along with the others are mainly focusing on achieving as low latency as possible combined with as high performance as possible[5].

There are other database solutions that focuses on supporting **C**onsistency and **P**artition tolerance, and partly supports Availability. Their partition tolerance may often be obtained by mirroring database clusters between different data centers. The main advantage is the possibility to achieve quicker response by splitting the workload into different sub tasks and then executes them simultaneously across all available nodes/servers[5]. The consistency level may be important for some systems like a stock market. The stock prices of a stock market and number of stocks available will always have to be up to date. It is the same principle for an e-commerce website - it would not be good for the business if the customer finds out the product is out of stock after he or she submitted the payment.

Even though different database systems are grouped in figure 3.1 on page 16, it does not mean this is always the case. The grouping is based on their default out-of-the-box setup. There are different needs for different situations, and there may be necessary to change the behavior for a database system. E.g., how much data to keep in memory before flushing to disk, strengthen the consistency level for a cluster, and so on. Figure 3.1 on page 16 is not the golden rule; it is just a visualization of the initial setup of the solutions, and their out-of-the-box support for the CAP theorem abilities.

3.3 NoSQL data stores

3.3.1 Extensible record stores

In Cattell's article there is a brief explanation of what extensible record stores are[9]. Cattell describes the data model of an Extensible record store to be almost identical to Google BigTable's data model since its design is made up of rows and columns, and its flexibility by splitting both rows and columns across multiple nodes when scaling. When data is split across multiple nodes, the data is stored and later retrieved based on a predefined key. Splitting data from the same key across multiple nodes is called *sharding*.

Even though column and rows may be split across multiple nodes throughout the cluster, the location of the data is not randomly selected. It is also possible to design the system so that data that is supposed to be retrieved together, e.g., a user's *username* and *password* is stored together by combining the columns into a column group. Column groups have to be predefined before storing data, as it is used to determine the location of the data. When two or more columns are located in the same *column group*, Cassandra will try to store all of them on the same node. Cassandra will even try to store them as close to each other on disk as possible to decrease the amount of time used to retrieve the data from disk. The reason why retrieving data located closer to each other physically on the disk is faster and does matter, is because how the operating system and the disk is constructed. The operating system will read *blocks* of data. A block of data

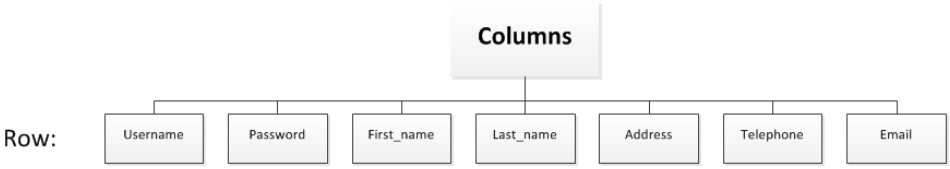


Figure 3.2: An example of a relational database row, consisting of multiple columns..

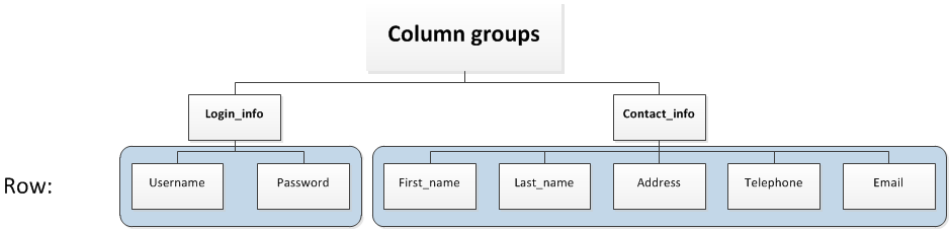


Figure 3.3: An example of how extensible record stores split rows and columns into column groups..

contains n number of bytes, which is a fixed size of bytes predefined by the OS. The OS will always retrieve at least one block of data, even though only one byte is needed. If it is possible to read all the data necessary as one stream, meaning that the hard drive does not have to change location of its reader head (the original mechanical hard drives) but may continue to read block by block after each other until all the data is retrieved, the reading time will be lowered, as moving the reader head takes time.

A traditional relational database row-column illustration with users password, username and contact information is seen in Figure 3.2. The figure displays a single row of data made up by seven columns. There is no way the database system is able to detect that e.g., *username-* and *password-column* are closer related than the *username-* and *address-column*. The relational database system takes for granted that all the columns are equally related no matter the position within the schema. The relational database system will also require all fields to have a value, even though it is null. It has to be allocated space for all columns whenever a row is inserted, even though only a few values are inserted. E.g., if a table consists of 10.000 columns and data is inserted, the database system must allocate space for all columns, even though not all columns have a value.

Extensible record stores are more flexible as space is not allocated for columns that do not have a value. As mentioned earlier, extensible record stores partition columns into column groups as shown in Figure 3.3. The figure visualizes the columns that are connected through column groups. The column groups have to be pre-defined before data is stored, since they are used to determine the storage-location of the data. They are also used to determine the storing-order to make sure grouped data is located physically close when it is flushed to disk. Even though the column groups must be defined before data is passed on to the extensible record store, the columns does not have to be defined as there does not exist any pre-defined

schema like the relational database systems have for the columns. One row may contain e.g., *username*, *password* and *firstname*, while another row may contain *username*, *password*, *firstname* and *lastname*. The first row is missing the last name column. This column does not contain a null-value, as it would have done in a relational database, it does not exist and therefore will not occupy disk-space. Even though most of the columns within the same column group will be located at the same node, it may happen that the data will be split across multiple nodes if there is lack of memory or disk-space.

The column group-implementation is a very clever and neat way to handle rows that may be hundreds or even thousands of columns wide. Even though all the columns within the row logically belong together at a very high abstraction level, it will most likely never be accessed at the same time if there are hundreds or thousands of columns. However parts of the row will "always" be accessed together and therefore allocated within the same column group. The data, which is not strongly connected, is located within different column groups, which tells Cassandra that it is okay to split the row across multiple nodes. Even though the links between columns within different column groups are weaker than the link between columns within the same column groups, they may still be accessed together as they are all subsets of the same data set.

Extensible record stores does not require the columns within a row to be stored sequentially enabling the possibility to extend column groups with new columns whenever needed, or omit columns if they are not needed for the current data set. When adding new rows to a relational database table, the whole schema has to be changed and each row has to extend the newly added row. When adding new rows and columns to an extensible record store, it will never affect the already existing data. Even if there are added columns to already existing rows, as the new columns will be appended to the end of the data set. The majority of the implementations are append-only implementations, meaning that the data which is written to disk, is always appended, no matter if it is an insert or update. Periodically, the appended data is read and matched against the already existing data set. If data that is currently added, also exist within the database, the database update its local values with the newest version.

Apache Cassandra is one of the most known extensible record stores today. Other examples of extensible record stores are *Apache HBase* and *Hypertable*.

3.3.2 Key-value stores

Key-value stores do not have the typical pre-defined schema as the relational database have[20]. A schema is a contract with a detail description of the tables in the database. All columns and their data types are also described for each table. This strict structure is replaced by a more relaxed storage structure where a key represents a set of data like mentioned for extensible record stores. Hewitt visualizes the data model of a typical key-value store as a bucket-like data model, were data sets are

dropped into a bucket. A bucket may contain many data sets, which may be *picked up* based on its key. A bucket is a very good representation as it is an unstructured container for objects and/or elements. In a relational database it is possible to retrieve a row by querying for any of the columns in the row. This is very powerful as it is not necessary to predefine which columns to index ahead of the insertion to be able to retrieve it again. For key-value stores, it is only possible to retrieve data based on the unique key that is assigned to each data set. The index is constructed based on the keys and therefore it is important to assign a meaningful key, as it will be used to retrieve the data later on. The key-value model gives the developers more responsibly of structuring the data in the database. This opens up for more possibilities when it comes to database-designs, although it puts more pressure upon the developer to carefully design the database. Since the rules for designing database models for relational databases are so strict, it is easier to succeed than when designing a database model for a key-value store. Since key-value stores does not apply the same strict rules for their database-design, the developers will have to be more careful while designing the database in order to make a working database model.

In his paper, Cattell tells us that a key-value store generally provides some sort of persistence mechanism[9]. The database implementation may be programmed to either store data directly to disk, which will strengthen the database-consistency, while the performance will be weakened. The performance will be weakened since the transactions will have to wait for the database to successfully write the inserted data to disk in order to finish. If every transaction will have to wait for the disk to return a message indicating that the write operation were successful or not, systems that is very write-intensive will suffer from poor performance and response time. As mentioned earlier, writing to disk is the most expensive operation for a computer today, at least if it is a mechanical hard-drive. The data may also be stored asynchronously to disk. This will weaken the consistency as it takes time writing to the disk. It will also take some time to transfer the data to other nodes in the cluster, and make sure that the next read initiated to any nodes within the cluster retrieves the newly written data. It will however strengthen the performance as the application(s) will not have to wait for a successful write-to-disk acknowledge message, and therefore may continue as soon as the data is sent to disk for storing. A few examples of the most known key-value stores are *Amazon's Dynamo*, *Project Voldemort* and *Redis*.

3.3.3 Document stores

Document stores are related to the daily term *document*. It may store documents like e.g., Word-, Excel- or PDF-files. The documents stored in a document store may also contain other documents, scalar values or lists. The attributes of each document are defined during runtime and therefore each document may contain different attributes even if they are of the same *type*[9]. Since there is no schema defining the structure of the documents stored, the document stores is a very flexible database as

practically everything may be stored. However it weakens the structure of the database, and if the design is not done properly, the inserted documents may be hard, or even impossible, to manage. Therefore it is even more important now, than with a relational database, to create a database design that is easy to understand and implement.

Document stores also support *secondary indexes*. A secondary index is just another key that may be assigned to the data set. It will be indexed and possibly used to lookup the data. Examples of known document stores are *Apache CouchDB*, *MongoDB* and *Riak*.

3.3.4 Graph databases

A graph database is a database that focuses on the *relations* between objects e.g., people. *Neo4J* is a well-known graph database, which is used by 20 of the top 2000 it-companies and hundreds of new startups all over the world[28]. Relational databases are the preferred data model for handling financial tasks, reporting etc. Key-value stores is designed for tasks like handling the shopping carts for an e-commerce site where the number of elements in the list is unknown, the object type is unknown and there may be high write throughput. While these data models covers most of the known areas, there are areas they do not cover and areas they are not designed for and where their performance will suffer; relationship between objects or data sets. Key-value stores or relational databases are not very good at representing relationships and connections between objects and/or people. Graph databases on the other hand are designed to focus on the relationships, e.g.:

A person and a car - The person may *own* the car, he may *rent* the car, *borrow* the car or he may even have *stolen* the car.

Between two people - They may be *friends*, *lovers*, *siblings* or *enemies*.

Conferences - Conference talks are often connected with people and stages/rooms as it is held *for* someone *at* a stage or in a room, there may be another speaker *after* the current speaker, and there is also a *speaker* for the current talk. This list may be expanded depending on what is interesting.

These are just three simple examples of what a graph database may be used for. Twitter uses a graph-database on top to connect tweets, your followers, who you are following etc.[15]. Twitter need to traverse a users followers as fast as possible to be able to deliver tweets as close to real-time as possible. Their choice of database had to be able to rapidly lookup connections between people, as well as handle high write throughput as new tweets are posted, people follow, and stop following each others etc.

3.3.5 Other known data models that are not covered

Object-oriented databases store objects, which is very similar to the objects a programmer is familiar with. *Distributed object-oriented databases* are

the same as Object-oriented databases, except that they distributes their objects across multiple nodes/servers and keep as many objects as possible in main memory to increase performance, as the response time will be lowered since there will be less disk-accesses.

To read more about these models, I recommend reading Cattell's paper about *Scalable databases and NoSQL Data stores*[9]. This thesis will not go into details of these data models.

3.3.6 NoSQL advantages and disadvantages

"NoSQL is a better choice than a relational database for the given task". Those who did not take the time to consider the advantages and disadvantages of NoSQL solutions may hear this statement. There are also many different NoSQL solutions that exist, and therefore it is not always a better choice than the traditional relational database. Whether or not a NoSQL solution is a better choice than the relational database, depends entirely on the system requirements for each individual system. In some cases, e.g., when the database is responsible for people's finances, we can not tolerate guessing or eventual consistency, which some NoSQL solutions provide to increase performance[5]. What would happen if the balance of a bank account were not accurate at any given time? If the financial systems were not consistent, it is very certain that the world's economy would have been affect in a way that is hard to predict. There should never be any doubt if the amount of money currently registered to an account is the actual amount or not. Transactions that are handling money transfers have to be ACID compliant in order to guarantee that the balance for both the sender and the receiver is adjusted accordingly whenever the transaction is complete.

If a system is not required to be ACID compliant, the world of NoSQL opens up. However, a few of the NoSQL implementations are ACID compliant e.g., Neo4J and CouchDB. If a system does not require the ACID guarantee to be applied, it is often because it was sacrificed in order to achieve a successful implementation. Applications like web analytic tools and social network feeds are not ACID compliant as it would ruin the performance, and make the application useless. A web analytics tool will most likely receive multiple streams of data to be recorded at the same time, depending on the traffic of the website(s) monitored. To prevent the analytics tool from loosing data or stacking up too much data before it is stored, the write performance of the implementation must be high. A traditional relational database is not able to process the huge amount of data received at the rate which is required for a web analytics tool to be functional at all times. The data model of a relational database is not flexible enough to handle the unknown number of different websites, recordings etc. which may be stored. The relational database is also generating a lot of overhead due to it being ACID compliant, which is not necessary when handling website statistics. In most cases it will not cause any problems if data is stored a second or so after its originally passed to the database, or if the stored data is not shown after a few seconds.

Performance gains

The biggest advantage of using a NoSQL database depending on the solution is the query speed, response time, fault tolerance and the scalability. Most NoSQL solutions perform better than traditional relational databases when measuring performance since most of them are not ACID-compliant. As mentioned earlier the ACID guarantee generates a huge overhead and is the bottleneck of the relational database systems. Each implementation that is following the CAP theorem is only able to strongly support two out of three sides (see Figure 3.1 on page 16)[20]. Since the implementations are not ACID compliant, they are able to achieve goals that are not possible with a relational database. Different goals may be achieved depending on which part of the CAP theorem the implementation supports.

One of the biggest advantages for NoSQL implementations that focus on ***P****artition tolerance* and ***A****vailability* is the ability to scale in real time without lowering the performance. Cassandra strongly supports Partition tolerance and Availability, while it partially supports consistency as it supports eventually (weak) consistency. Cassandra offers the ability to replicate data across multiple nodes to keep availability high. If one node goes down which happens from time to time, the data will still be available as it is replicated to n other nodes. By increasing the replication factor, Cassandra also increases the availability. However there are some limits: If the replication factor is 2, the total size of the cluster will be twice the size of the actual data, if the replication factor is 3, the total size will be 3 times the actual data etc. Therefore database administrators and developers have to find a balance between what is necessary and what is feasible. Cassandra's replication factor may be customized for every keyspace, at every data center. The `datacenters.properties` file sets the replication factor for Cassandra.

The replication of data across multiple nodes happens asynchronously to prevent performance loss. If a transaction has to wait for each node to successfully store and return a success-message, the benefit of NoSQL would be lost. When accessing data from a node which stores replicated data of the desired data set, it may happen that the data retrieved is old as the newly inserted data is not replicated to the actual node yet, or the node have not yet stored and compacted its data. This is why Cassandra's consistency is called *eventual consistency*, as it will eventually be consistent whenever data is replicated and made available to all nodes that are responsible for the given data set.

Main disadvantage

The biggest drawback for most NoSQL solutions is that they are no longer ACID compliant. The ACID guarantee provides a well-known and tested transaction security, which is extremely valuable in some cases. When a database transaction is ACID compliant, it is ensured that the data stored is the same data retrieved by anyone accessing the database at any time

after the insertion. This is not always the case for the NoSQL solutions, which replicates its data to other nodes asynchronously.

NoSQL solutions will in most cases also require more disk-space and computing-power than relational databases. Replicas of the data are stored, which is why the database will require more disk-space than the actual data stored. The NoSQL solutions are also used for storing larger amounts of data than the relational databases as well. E.g., web analytics data, social network feeds, and search-engine indexes.

When will a relational database management system be a better choice?

If the requirements are not thousands of reads and/or writes per second, they do not consist of an incredibly large amount of data covering tens-, hundreds-, or even thousands of columns, or the response time of the system has to be extremely low, a traditional relational database like MySQL or PostgreSQL may be a better choice than any NoSQL data model currently available. The relational database model is well known and developed. There are a lot of people who knows how to work with the relational database model, and it provides well known functionality like the traditional SQL query language, and the ACID guarantee for its transactions. If the application(s) using the database does not need anything else than what the traditional relational database may provide, there is no need implementing anything else either. As mentioned earlier, it is easier to get competent developers for relational databases as its a well known data model, as well as the comfort in a data model that have been around for 20+ years, used by all kinds of applications and still are the preferred database model in many cases.

Chapter 4

Cassandra

4.1 Introduction

Lakshman and Malik, two engineers at Facebook, designed Cassandra and open sourced it in 2008 to the Apache community. They describe Cassandra as a *"distributed storage system for managing very large amounts of structured data spread across many commodity servers, while providing highly available service with no single point of failure"*[23].

Facebook needed a storage structure that could solve their *Inbox search problem*. Inbox search is a feature Facebook developed to let users search through their inbox recursively[23]. To keep latency as low as possible, Facebook needed a solution that not only were able to distribute data across data centers with different geographical locations, but also between nodes within the same data centers. Facebook began to develop Cassandra, their solution to the problem, which was inspired by Amazon Dynamo[23]. Even though Cassandra is very similar to Amazon's Dynamo that is used for Amazon's shopping-cart feature, there are some differences as Cassandra was designed to solve different problems than Dynamo. For every write operation made to a Dynamo database, a read operation will be required as well. This would be very limiting for the kind of system Facebook were developing since it is a very write intensive system. A huge advantage for Facebook, and probably one of the reasons why Cassandra is very similar to Dynamo, is that one of its two engineers, Lakshman, were one of the authors of the Amazon Dynamo-paper[14].

Cassandra is a key-value store, which means that it has a key connected with every set of data. The key is used to identify the data set when the data is retrieved. Since the key is the only thing to identify the connected data, it may happen that there will be stored duplicates of data like postal codes in the database. This is one of the downsides of the design, although the flexibility, availability and IO-speed compensate very well. Even though there will be stored some duplicates, it is normally not an issue as disk space has become cheaper. Guesses are made that todays database systems never uses all of their disk space, as the disk space often is extended whenever needed to an *"infinite"* number of Megabytes.

The attributes that is often accessed together from a data set and

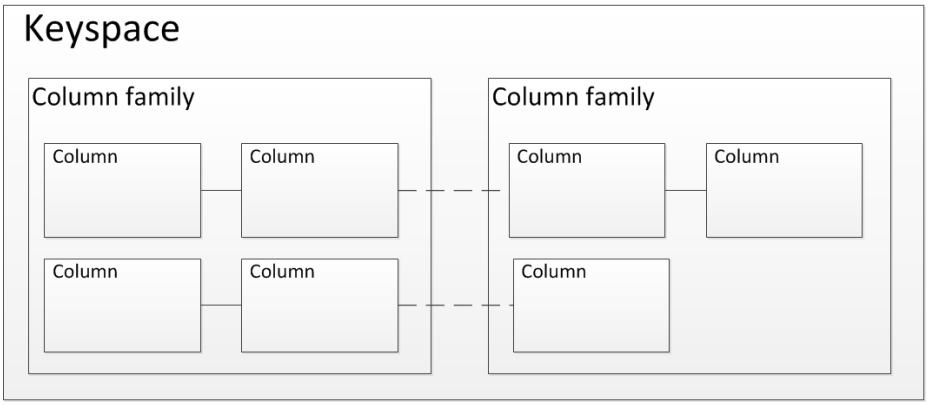


Figure 4.1: Cassandra's data-model

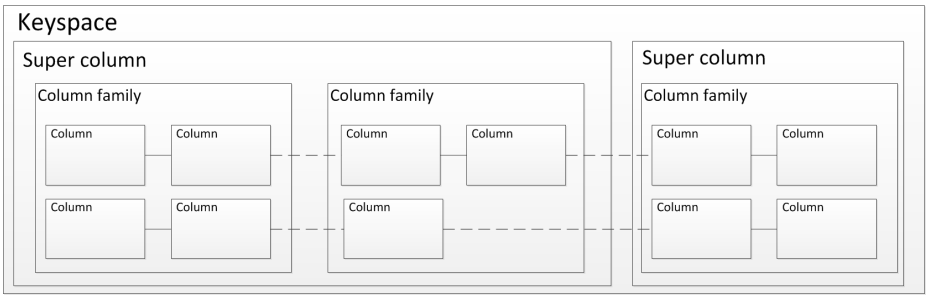


Figure 4.2: Cassandra's data-model with SuperColumns

logically "belongs together" based on the application design should be grouped together in *column groups*. Cassandra tries to store data from the same column group at the same node, and preferably as close as possible on disk.

4.2 Data model

The official Cassandra wiki describes Cassandra's data model saying it is *"designed for distributed data on a very large scale"*[12]. Cassandra operates in main memory, periodically performs asynchronous storing-operations to disk and got rid of the ACID guarantee to increase performance and availability. The structure of Cassandra's data model is quite different from the traditional relational databases' data model. Figure 4.1 displays the simplest version of its data model. The highest level is called *keyspace*. Usually each application has its own keyspace. A keyspace may be compared with the relational database model. A Cassandra cluster may consist of multiple keyspaces, which makes it possible for multiple applications to operate on the same Cassandra cluster.

Below the keyspace-level there is a level of *Column families*. A column family is a set of one or more columns that is logically grouped together based on the database design. Column families are applied as a wrapper to prevent data often accessed together to be physically located too far apart

from each other. Keeping related data physical close to each other increases the performance. This will lower the lookup-time, as there will be less disk-reads and data accessed from multiple nodes. If the column family abstraction is not enough, it is possible to group column families into *super columns*. Super columns contain one or more column families and are just another abstraction like the column families. Super columns are not always used, but there may be cases were e.g., there is a wide range of data stored that they may be needed. If the super columns are present, the data model will look like figure 4.2 on the facing page

Within a column family there will be one or more *column(s)*. A column is the lowest data-structure within the Cassandra data model. A visualization of the column data structure is seen in figure 4.3 on the next page. A column consists of three attributes: *name*, *value* and *timestamp*. The name represents the column name and is used to identify the column. When storing data about a user, the names may be: name, email, address, etc. The value attribute contains the stored value. The timestamp is the actual time when the column was initially stored. It is used when Cassandra replicates data across multiple nodes and the actual column already exists for the other node(s). If the column already exists, Cassandra will compare the timestamp of the already existing column with the newly retrieved column and keep the column with the newest timestamp. To successfully be able to compare timestamps the systems have to be synchronized so the timestamps will be accurate without taking into consideration where the servers are located. Since the nodes often may be located in different time zones the timestamp should be converted into an universal time zone before applied to the column. Each node should then retrieve its current time and convert it into a known time zone, e.g., UTC. After the timestamp is converted into the universal time zone, it may be applied to the column before its stored and replicated.

A *"row"* may be compared with a relational database row, as it is a set of values connected together. However, there are some differences between the traditional relational database row and a Cassandra row. The relational database row is static as the number of columns is final, while a Cassandra row is very dynamic and the number of columns may vary. One row may contain e.g., 10 columns, the next contains 5, while the last contains 100 columns. The flexibility of what is possible to store and the idea that there are not allocated space for columns which are not part of the current data set is one of the beauties that Cassandra offers. However, with flexibility comes responsibility for the developer. Since there are no strict rules for which columns to be stored, it is the developer who decides how to structure the data that is stored. If this is not done properly, the database may easily become chaotic, and finding what you are looking for may be hard. If the data is poorly indexed, the database may become useless. Distributed database systems with enormous amounts of data do require a well-structured index in order to provide good performance.

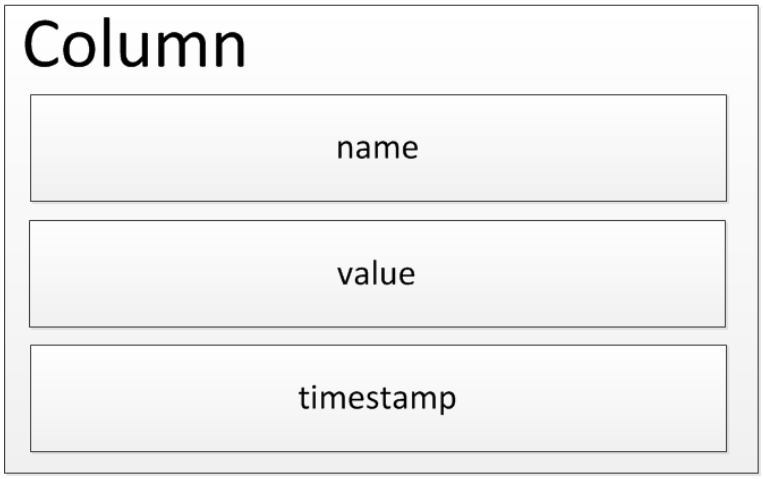


Figure 4.3: Content of a Cassandra-column

```
keyA:
    keyB: value
    keyC: value
keyD: value
```

Figure 4.4: Yaml-file example

4.3 Node and cluster configuration

The parameters used to configure nodes in a Cassandra cluster are separated from the compiled code in a configuration file named *cassandra.yaml*. When configuration parameters are moved to a separate file it is possible to tune the implementation without having to recompile the source code. Cassandra uses a file-structure called *yaml*, which is an acronym for *"Yet another markup language"*. The yaml-structure is a key-value structure that also supports nested values as seen in figure 4.4.

The configuration-file is loaded into memory upon startup by a class called `DatabaseDescriptor`. As mentioned earlier the attributes found in the configuration-file may be changed depending on the needs, and by restarting the application the new attributes will be read from the file and used. Some of the attributes found in the configuration-file that is relevant for this thesis are:

- *cluster_name* - The cluster name the node is supposed to interact with. Each cluster should have its own name to make sure each Cassandra-instance interacts with the correct cluster in case multiple clusters are running on the same server.
- *initial_token* - If empty the node will request a token from the cluster which will assign the node 50% of the token range from the most busy node in the cluster. If no load information is currently available e.g., when a new cluster is initialized, the node will be assigned a random

token. As the tokens are assigned randomly there is a chance of the cluster being unbalanced as there are no data available to calculate the most busy token range or calculating tokens so that there will be an even distribution of data across the nodes.

- *partitioner* - Define the partition to use for distributing rows across nodes in the cluster based on the key. Custom partitioners may also be used as long as they exist on the class path and implements the `IPartitioner` interface. Out of the box Cassandra provides three partitioners: *Murmur3Partitioner*, *RandomPartitioner* and *ByteOrdedPartitioner*.

    1. RandomPartitioner distribute rows evenly by hashing the keys using the md5 algorithm.
    2. Murmur3Partitioner is similar to RandomPartitioner except that it uses another hashing algorithm named *Murmur3_128*.
    3. ByteOrdedPartitioner order the rows by an ascending order, based on the keys. This allows scanning of rows in key order since the rows will always be sorted.

- *data_file_directories* - Cassandra's data-directory. Make sure Cassandra is able to read and write this folder.

- *commitlog_directory* - Cassandra will store all commit logs in this folder. Commit logs receives all data written to the Cassandra database and in case of restart, the commit logs are read to reload data which are not yet flushed to disk. When data is flushed to disk it will also be removed from the commit logs.

- *listen_address* - The IP-address the other Cassandra nodes in the cluster initiate connection with in order to interact with the node. If this attribute is left blank the Cassandra-implementation will try to retrieve the IP-address by using the *InetAddress* implementation, which in some cases may be wrong (e.g., if behind a router and the internal IP-address is returned instead of the external IP-address).

For the automatic scaling implementation it will be necessary to monitor system variables like the CPU-, memory-, and disk-usage over a timespan of n seconds. All these thresholds and the number of seconds the system has to breach the threshold(s) in order to send a scaling message back to the master will be extracted out of the implementation. They will be put into a configuration file so they may easily be accessed and to avoid recompilation of the source code.

4.4 The gossip protocol

The gossip protocol is a communication protocol often used by modern distributed systems that includes thousands of nodes. The gossip protocol is inspired by the traditional definition of gossiping with a little twist

where the members meet e.g., every nth hour. The first time A tells B something. The next time A tells C and B tells D the same thing A told B last time etc. This strategy leads to twice as many people knowing what have been told after each meeting. Although it is almost like normal gossiping, the content of the information shared will always be the same compared to what may happened during traditional gossiping between people. The reason why the information changes is because people often tells the story a bit different each time, and after the story has passed on to n number of people, the information initially shared may be something completely different than what is currently being shared. The gossip implementation initiates connection against another random node that it will share information with. The node will share information about itself and information retrieved from earlier gossips. By sharing information retrieved from earlier gossips as well, the information spread fast throughout large scaled clusters as gossip between two nodes only shares a small amount of data, and there are multiple nodes gossiping at once.

The gossip protocol is very robust since the same information may be passed on from many different nodes. Node A will choose randomly node B in the cluster to pair up with, and share information about itself and about others. If node B are struggling with e.g., hardware failures or are not available for some time, other nodes will pass on their information, as long as it managed to at least send out the information to one node. If node A pairs up with node C, which are not responding at all for a certain amount of time, it will be marked as *down* and share this information with other alive nodes in the cluster.

Cassandra is a decentralized system meaning it has no master or single point of entry, and any node in the cluster is a potential access point. If an incoming read-/write-request is meant for another node, the node will forward the request to the correct node. The gossip protocol is very well suited for a decentralized system as the communication is initiated from any random node at any given time to another random node. Since Cassandra does not have a single point of entry, it means it also does not have a single point of failure. If one node goes down, the database and its data will still be accessible as a connection may be initiated to any other node.

The *Gossiper-class* implements the gossip protocol. The gossiper manages the message sent and received, and keep lists of currently live- and dead nodes. A gossip between two nodes is a three-phase communication, which is periodically triggered every second [20]:

1. Node A sends node B a `GossipDigestSyn`-message. The message contains the name of the cluster, the name of the currently used partitioner and a list of endpoints (nodes) and the largest state they have generated known to the node sending the message. The number of generated states is used to determine which message is newest if received more than one message containing information about the same node.

2. Node B returns a `GossipDigestAck`-message to node A. The message is a response to the `GissipDigestSyn`-message and contains the same type of list for known nodes as received from node A, except that the list contains information about the nodes known by node B. It also contains a map with the current state of each endpoint (node). The state indicates if the node is alive or not and may contain state-information about which data center the node is located in, which rack, its internal IP-address, etc.

3. Node A sends a `GossipDigestAck2-message` to node B to complete the gossip-round. This message contains the same map as received by the `GossipDigestAck`-message, except it also contains the information known to node A.

4.4.1 Hinted handoffs

Hinted handoffs are implemented as an extension to the gossip protocol. They are initiated if data is written to the cluster through node A when it is originally supposed to be located at node B, and node B is not currently available. Since node B is not available, node A have to temporarily store the data as a side-note in order to pass it onto node B whenever it comes back up.

In most cases this is a good idea. It prevents the current write-operation to be put on hold until the responsible node comes back up, and it prevents loss of data. However, there are some practical problems with hinted handoffs as well. If a node have been dead for a while and suddenly comes back online, all nodes who currently holds hinted handoffs for the node will start streaming data to the node. This may lead to multiple large streams of data received at the same time, which may overwhelm the node at the most vulnerable time. The most vulnerable time of a node is just after it is attached to the cluster/ring and struggles to learn the *topology* of the ring. The topology is the top-level structure of a Cassandra cluster. It consists of the data center(s), racks and nodes, which also is the physical structure of the cluster. The topology also describes which racks that exists within each data center, and which nodes that is within each rack. A line from the topology property-file may look like this: `192.168.0.1:NYC:RAC33`. This line tells us that the node `192.168.0.1` is located within the `NYC` data center at rack `RAC33`.

Cassandra provides the possibility to either turn off hinted handoffs completely or reduce the priority to solve the problem[20]. If hinted handoffs is turned off, the risk of loosing data is heighten, as there is a change that all nodes containing the written data is down. Therefore it is recommended to lower the priority of the hinted handoff, allowing more important operations to finish before information is streamed to the responsible node.

4.5 Merkle tree

A Merkle tree is a hash tree that contains a summary of the data set. Cassandra uses Merkle trees during major compactions to avoid sending unnecessary data across the network. Leaf nodes represent unique datasets, while nodes further up in the tree represents subsets of the actual node's data. The merkle-tree implementation provides Cassandra with the ability to check one branch of the tree at a time without having to download the entire data-set as one branch represents the hashed values for one key including its subsets.

4.6 Stages

Cassandra's lifecycle is split into multiple *stages*[38]. Each stage is responsible for its own area, e.g., the *gossip stage* is responsible for keeping track of which nodes is dead or alive in the current cluster as describe in section 4.4 on page 31. The gossip-stage is *single-threaded*, as there will only be one gossiper running at a time. Even though the gossip-stage is single-threaded, it is not a potential bottleneck if multiple nodes communicate with the same node at once. The stage continues as soon as a message is sent or received, meaning that it does not wait for an answer. This allows node B, C, and D to start gossiping with node A, even though node D is already gossiping with node A.

The available stages are defined through an enumerator called *Stage*. The Stage- enumerator also contains information if the stage is an *internal*- or *request*-stage. An internal-stage is a stage that is performed locally within the actual node, where other nodes of the cluster are not involved e.g., flushing local data from memory to disk. Request-stages are stages requesting information from other nodes within the cluster.

Regardless if the stage is an internal- or request stage, it is either *single*- or *multi-threaded*.

4.6.1 Single-threaded stages

Single-threaded stages run on a single thread, which means that there will never be more than one instance running at the same time. The single-threaded stages currently implemented are:

- **Stream** - The stream stage is initiated when data is moved/streamed between two nodes in the cluster.
- **Gossip** - The gossip stage is initiated every second from the *Gossiper*-class. The Gossiper will connect with a random node in the cluster and share known status updates about dead and alive nodes.
- **Anti-entropy** - The anti-entropy stage matches replicas of data-chunks against each other and updates all data chunks to the newest version currently available. Cassandra's anti-entropy implementation is based on Dynamo's, which uses a *merkle tree* (see section 4.5).

- **Migration** - The migration stage is initiated when data is migrated from one node to another. It will initiate the stream-stage when moving data. When the data is successfully moved to the new node, the stage will remove the data from the initial node.
- **Read-repair** - The read-repair stage is initiated each time a read is requested. It will be running in the background comparing all available replicas of each data set to determine which one is the newest.
- **Tracing** - The tracing stage is tracing sessions e.g., a users query throughout the Cassandra-instance. The stage will record all local and external events that happen during the current session.
- **Misc** - The misc stage is all operations that do not match any other stages. E.g., stream-replies replication-finished and snapshot messages retrieved.

4.6.2 Multi-threaded stages

Multi-threaded stages are stages that there may be multiple instance of running at the same time. During the initial startup of Cassandra, these stages will be assigned n number of threads through a *thread pool*. The number of available threads is defined by the `concurrent_reads` parameter in the configuration file. A thread pool is a collection of threads that is currently not in use. A multi-threaded stage may collect an available thread from the pool to initiate the staging process on. The currently implemented multi-threaded stages are *Mutation*, *Read*, *Request-response*, *Internal-response* and *Replicate on write*. The reason why stages like the read-stage are multi-threaded is because Cassandra is supposed to handle multiple reads at a time to keep performance as good as possible. If the read-stage were single threaded each new read had to wait for the previous read operation to finish before it could access the database. Since there may potentially be hundreds or even thousands of connections at the same time, there cannot be only one stage active at a time responsible for reading data from the database. It will weaken the consistency as there is no guarantee all the read-stages will retrieve the newest version of the data at the same time. However, Cassandra only supports eventual consistency, which means that the data will be consistent at some point.

- **Mutation** - The mutation stage is activated when a node is currently recording hinted handoffs - data received which is intended for another node that is currently not available.
- **Read** - The read stage is initiated each time a read operation is performed.
- **Request-response** - The request-response stage is whenever the current node performs a response to a request received from an external node.

- **Internal-response** - The internal-response stage is whenever the current node performs a respond to a request received from an internal operation.
- **Replicate on write** - The replicate on write stage is initiated when data is written to the database. It sends messages to the nodes that will be replicating the data to initiate a writing-operation as well.

4.7 NodeTool

Nodetool is a tool built into Cassandra which provides a simple command line utility to interact with Cassandra to monitor the node- and cluster status as well as managing the cluster by e.g., moving nodes to solve hotspot-issues[31]. To interact with Cassandra through the nodetool interface, execute the `bin/nodetool -host <IP> -p <port>` command. If no port is provided, nodetool will connect to the host-address through the default port *(8080)*. Some of the commands available from the nodetool interface found at the official Cassandra Wiki are listed below[31]:

- **Ring** - To get an overview of the Cassandra cluster. It tells which nodes are currently alive and dead, how much storage used for each node and if the data is evenly distributed or not.
- **Join** - Tells the connected node to join the ring. It assumes that the node was initially started with the *-Dcassandra.join_ring=false* flag set as it prevents the node from automatically joining the ring.
- **Info** - Returns detailed information about the connected node. The information returned contains e.g., the node's token, how many bytes currently stored on the node, the current uptime of the node in seconds and its current memory usage.
- **Cleanup** - Remove tokens that no longer belong to the connected node. This may be a token assigned to another node, or tokens that are not active anymore, as the node has been moved.
- **Decommission** - Decommission is the opposite of bootstrap which is the initial phase during startup. Whenever the decommission command is initiated the node is instructed to move its data somewhere else. The communication protocols are shutdown to prevent new data being written to the node. It may be used before removing the node from the cluster as it will mark the as being shutdown, which will prevent other nodes from generating hinted handoffs and waiting for it to come back up.
- **Drain** - Stops all writing operations and preventing new data to be written to the node while it flushes the data from memory to disk. Even though the write operations are stopped it is still possible to read data from the node.

- **Flush** - Flush the memory tables to disk without stopping either reading- or writing operations.
- **Removetoken** -Removes the node that the provided token is assigned to from the ring/cluster. The command cannot be initiated directly to the node, as it should already be dead. Since the node is dead it is not possible to initiate commands through the nodetool interface and therefore the command should be initiated to one of the other active nodes in the cluster.

Chapter 5

Related work

5.1 Netflix's Priam

Priam is a tool developed by Netflix to run alongside Cassandra. It automates some of the database administrator tasks like automatic token assignment and backup & recovery. Priam is actively developed by Netflix and has been in use since the middle of 2011. The current implementation of Priam only works for *Amazon Web Services*.

5.1.1 Amazon Web Services

Amazon Web Services (AWS) is a service offered by Amazon were anyone are able to rent anything from physical resources like CPU power, RAM and disk-space, to applications like a database, a blog, a social network, etc.[4]. The key feature of *cloud computing*[1] is that there are no up-front costs by investing in large data centers, hire infrastructure engineers etc. The capacity rented in the *cloud* may easily be expanded if e.g., the traffic or the amount of data suddenly increases, or even scaled down if not needed anymore. Since the cloud offers a rent-what-you-need model, it is perfectly suited for e.g., research project where researchers will need a lot of computing power over a short amount of time.

5.1.2 Netflix's motivation

Since Netflix moved their infrastructure from servers operated by themselves and onto the Amazon cloud in 2010, they were in need of a tool for managing configuration, provide backup & recovery functionality and automate token generation both within the same region, but also across regions. To solve this problem, Netflix developed Priam which they open sourced in February 2012[3, 1].

[1]Cloud computing is a very broad term that refers to services provided over the Internet[2]. It may be both renting a simple web server for hosting your blog to renting hardware in order to produce resource results. Cloud computing enables small and medium sized companies to produce amazing results, as they do not have to invest in expensive computer equipment upfront, but rather pay for what they use instead.

To have a reliable and stable backup solution is critical whenever a third party is involved in the process of delivering your services. Therefore Priam generates a snapshot each day, which is stored to amazons S3 storage system. Amazons S3 provides a very simple API and allows the application to access unlimited amounts of data from any of Netflix's nodes at any time[1].

Priam uses another Cassandra instance or a SimpleDB-instance to store meta-data about the nodes, clusters and regions. This information is later on used when assigning tokens to newly added or moved nodes. New nodes are added whenever a node fails, which will happen when dealing with a large number of nodes. Netflix uses an implementation called *Chaos Monkey* to create failures on their live clusters to test Priam[1].

Chaos Monkey

Chaos Monkey is an implementation developed by Netflix to generate failures within their Cassandra clusters[11]. It is open-sourced and may work on other services running on Amazons Web Services as well. Chaos Monkey seeks out groups of virtual machines, which is grouped together in an *Auto scaling group*[2]. Chaos Monkey is scheduled to run at low-peak times, and not during holidays. In most cases the auto-scaling feature solves the failure by itself, although it may happened that an engineer has to be involved. In these cases, Netflix want to be sure that the response time is as low as possible, therefore the tests are only set to run whenever there are people around.

5.1.3 Why we did not choose Priam

Priam could be used instead of creating another automatic scaling implementation from scratch. However, Priam is only able to run on the Amazon Web services. The implementation from this thesis is meant to be as flexible as possible. It should be able to run on Amazon Web Services, but it should also be able to run on other cloud services if desired.

Another reason for not using Priam is that Priam does not support an automated downscaling process like our implementation will do. There is a reason for Priam to not support automated downscaling- *"Scale up early, scale down slowly"*[3]. Netflix would like to be ahead of a possible performance loss by scaling up whenever breaching a certain resource cap for n amount of seconds. Our implementation will implement the same feature, but also have thresholds and timers for scaling down. Priam is not able to scale down since Netflix wants to be sure that downscaling happen slowly. Netflix wants downscaling to happen slowly to prevent removing capacity too fast or reduce an incorrect amount of capacity. If too much capacity is removed the cluster will be running on less resources than it

[2]Amazon offers an automatic scaling implementation for services running in the cloud, which is called Auto scaling group. It groups up virtual machines that will be monitored. If one of the virtual machines within a group goes down, the auto scaling group service should bring up a new one.

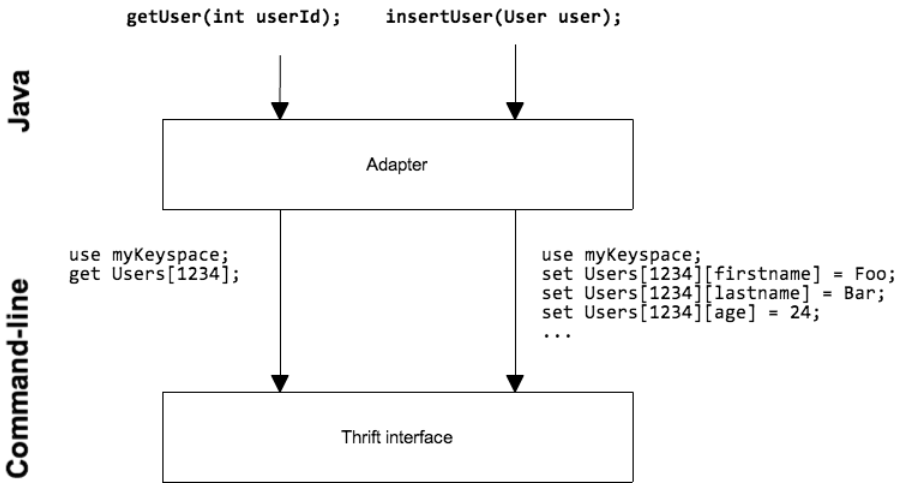


Figure 5.1: Abstraction of the thrift interface

should. When the cluster is running on less resources than it should, the whole cluster will be slowed down and in worst case data may be lost. Our implementation should find a way to automatically solve this problem.

5.2 Hector

Hector is a high-level client interface for communicating with Cassandra[18]. Cassandra provides a thrift interface that is command-based to insert or retrieve data. It also provides a command-based interface for cluster-, and node managing. Hector encapsulates all this into an abstraction layer that makes it easier to perform insertions/deletions against a Cassandra cluster. It also makes it easier to add or remove nodes, and retrieve cluster information.

5.2.1 Motivation

The provided tools that comes with each Cassandra-instance are pretty low level compared to the Cassandra implementation that is written in Java. The advantage of including low level tools which is command-based is that they may be used by any application written in any language by implement some kind of adaptor between the application and the thrift-interface provided as default by Cassandra. Figure 5.1 shows how a simple insert and retrieve may be abstracted through a java-interface.

Communication through the thrift interface

The `insertUser(User user)` method takes a User-object as input parameter and the adapter converts the User-object into commands, which is passed onto the command-line. Each at- tribute is represented by a command-line were the user-id is the unique key representing the row,

and the attributes name (first name, last name, age etc.) is used as column-name. The adapter also needs to know which keyspace to operate on before insertion. The keyspace is known initially when connecting to the database, as there is often one keyspace per application. The developers have already designed the database for the applications needs, and are therefore able to provide the keyspace necessary to access the data.

An application may also want to retrieve the data stored for a specific user through the `getUser(int userId)` method. The method should connect to the correct key space and ask for a user with `userId` as key. If data is found for the given key, the thrift interface will return all columns associated, one column per line. Each column contains the column name, the value and the timestamp of the retrieved column as seen in figure 4.3 on page 30. The Adapter will have to read line-by-line and insert the values into a User-object. The User-object will be returned when the adapter is done extracting values from the stream and into the User-object.

Current state of the cluster

Hector also has the ability to retrieve the current state of the cluster. It uses *nodetool* to retrieve information about which nodes are currently in the cluster and creates an abstraction-layer through objects representing each node. Since the nodes are represented as java objects, they are directly accessible by any java-application implementing Hector. Since hector brings the whole Cassandra cluster into java objects, it creates a perfect foundation for our implementation.

5.2.2 Why we did not use Hector

Hector provides access to any Cassandra cluster through a simple java interface. The reason why Hector is not used is that our implementation is supposed to work independently of the database software. Our implementation is intended to work for any types of cluster, not just Cassandra clusters.

Part III

The project

Chapter 6

Introduction

This part gives a detailed description of the project. It contains the goals of the project, and the design and implementation details. It also describes the limitations of the current implementation, and the failed implementation attempts that have consumed a lot of the time that were available for the thesis work. This part also explains the reason why the implementation has been named Hecuba.

6.1 Naming the implementation Hecuba

Our implementation is named after Cassandra's mother, Hecuba[19]. The Facebook team that designed Cassandra, and open sourced it to the Apache community in 2008, named it after the Greek prophet Cassandra[31]. Cassandra was the daughter of King Priam and Queen Hecuba of Troy[8]. Netflix, which uses Cassandra as their backbone for their video- streaming service, have developed a solution called *Priam*. Priam was Cassandra's father from the Greek mythology[32, 33]. There is also a Java-application that abstracts the communication with Cassandra that is named *Hector*[18]. Hector was Cassandra's brother from the Greek mythology.

Since her father, and one of her brothers names was chosen to create applications that were extensions to Cassandra, the automatic scale implementation should follow in the same footsteps. Our implementation was named Hecuba, since her mothers name was not used. A mother is someone who looks after her children, checking their health from time to time, putting them to sleep at night, and waking them up in the morning. A mother may be a very good representation for what the automatic scaler implementation is supposed to be for the Cassandra cluster it monitors. It is supposed to monitor the disk-, CPU- and memory usage of each individual node closely, and depending on the status message received, either scale up, or scale down the node(s). Our implementation will be running in the background without lowering the overall performance of the cluster.

Chapter 7

Goals and methodology

7.1 Goals

The goal of this thesis is to implement an automatic scaler for Cassandra clusters. Hopefully it will be possible to make a generic implementation, which may successfully be deployed to any type of cluster, not just Cassandra clusters. The implementation should monitor the resource usage of each node within the cluster separately, and scale up, or down if needed. It should not impact the overall performance of the cluster, meaning that the resource usage should not increase more than necessary in order for the implementation to run. The implementation cannot be deployed to the cluster if the resource usage increases, and the overall performance are lowered due to the implementation running. If this happens, the implementation has failed, and the design would have to be rewritten before new tests are performed.

7.2 Methodology

7.2.1 Kanban

Kanban has been used to keep control and progress of the thesis work. Kanban is an agile development methodology that is similar to *Scrum*[6, 34]. The biggest difference between Kanban and Scrum is that Kanban focus on *visualizing* the workflow.

Scrum consists of two lists, a *product backlog* and a *sprint backlog*. The product backlog contains all tasks that have to be done throughout the project. This list may potentially be very large, and the project manager will extract a portion of these tasks to create the sprint backlog of the current iteration. The sprint backlog contains the tasks that have to be done during the current iteration, and the tasks are taken from the product backlog. The lists should be prioritized in order for the developers to know which tasks they should pick first.

Unlike Scrum's product-, and sprint backlog, Kanban let developers create *columns*. Each column represents a stage in the development cycle. Figure 7.1 on the next page is an example of a Kanban board taken from

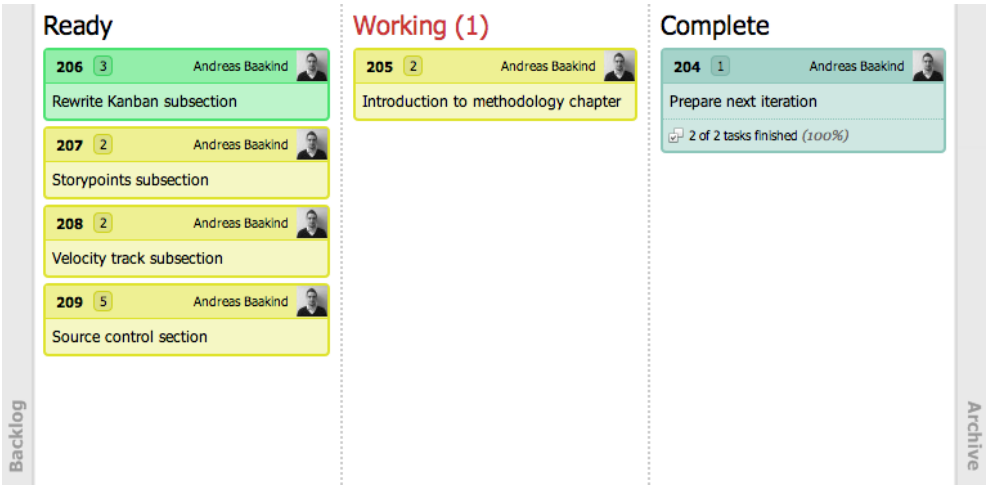


Figure 7.1: Kanban board example. *(Screenshot: http://www.agilezen.com/)*

the thesis work for this chapter. It has a *backlog* that is shown on the left side. The backlog is identical to Scrum's product backlog. The backlog consists of all tasks that is created, but not yet ready for development. The Kanban board consists of three columns in our example; *Ready*, *Working*, and *Complete*. Most of the time, Kanban does not use iterations. Instead, Kanban let developers continuously obtain tasks from the ready-column. A project manager should be responsible for feeding the ready-column with tasks from the backlog. However, it is possible to use Kanban and iterations, which I did during the thesis work.

The ready column is the column where tasks are located at the beginning of the current iterations. They should, but does not have to, be placed in a prioritized order so developers know which tasks they should pick. When a developer starts working on a task, he or she moves the task to the working-column.

The working column consists of tasks that are currently being worked on. The amount of tasks in the working column should be limited, based on the number of developers. E.g., during the thesis work when there was only one working on the project, the limit was set to 1. A limit of one task was set to make sure that all tasks that were started was finished before proceeding to the next task.

The complete column is the column where tasks that is completed is located. Each time a developer completes a task, it is moved to the complete column. Whenever there is time, the development team goes through the completed tasks. It may also be a dedicated test-team who approve or deny the completed tasks. If a task satisfy the expectations, it is are moved to the *archive*, or else it is moved back to the ready column. The archive consists of all tasks that have been worked on in earlier iterations.

The Kanban board shown in figure 7.1 is just an example. The board

may be adapted to any project, not just software development projects. It may also be extended with more columns to include e.g., a testing phase. There are no rules that define the number of columns, how to prioritize the tasks, task colors etc.. However, it is recommended to keep the most important tasks on top. Kanban has fewer restrictions than Scrum, which makes Kanban more flexible.

7.2.2 Story points

A number shown in the upper-left corner of each task indexes the tasks seen in figure 7.1 on the preceding page. On the right side of the index the estimated size of the task is shown. Each time a new iteration begins, the team goes together and estimates the size of each task. The size may be estimated in hours, days, story points etc. Since there is often hard to determine exactly how many hours a task requires, software developers tend to use a measure called *story points*. Story points are a measure used to indicate the workload of the tasks. Most frequently used are the Fibonacci numbers[37]: $1, 2, 3, 5, 8, 13, 21, 34, 45$, etc. The reason why the Fibonacci numbers are used is to create an easy-to-use scale for all developers. Computer programmers are often bad at estimating how much time it takes to complete a task, especially if all numbers from e.g., 1 to 20 are used. It would be hard to determine if a task should be estimated to 14, 15 or 16. Using the Fibonacci numbers where the distance between the numbers increases eliminates the problem. E.g., if a task's workload is estimated, and the team agrees that the workload required is more than 5 point, it will be estimated to 8 points, as this is the next number in the Fibonacci sequence.

After estimating tasks for a while using story points, the estimated values will be reflecting the actual workload better since it takes some time to get used to. It takes some time to figure out how much time that is required to complete 1 story point. The workload that is required for each story point may also vary from project to project, while working-hours does not. This gives the story points an advantage as they may adapt to any project of any type without adjusting the number-range used.

7.2.3 Velocity track

Velocity is often used in agile software development, and is calculated by summarizing the size of all completed tasks from the current iteration. The velocity is calculated after the iteration is completed in order to find out how much work that has been done. The velocity may be added to a list of previously calculated velocities to keep the *velocity track* of the development.

A graph showing the velocity track of the thesis work is shown in figure 7.2 on the following page. The graph shows the velocity track from August 17, 2012 until April 12, 2013, and each column represents the average story points completed during each iteration. The reason why the graph does not show the total amount of story points completed each

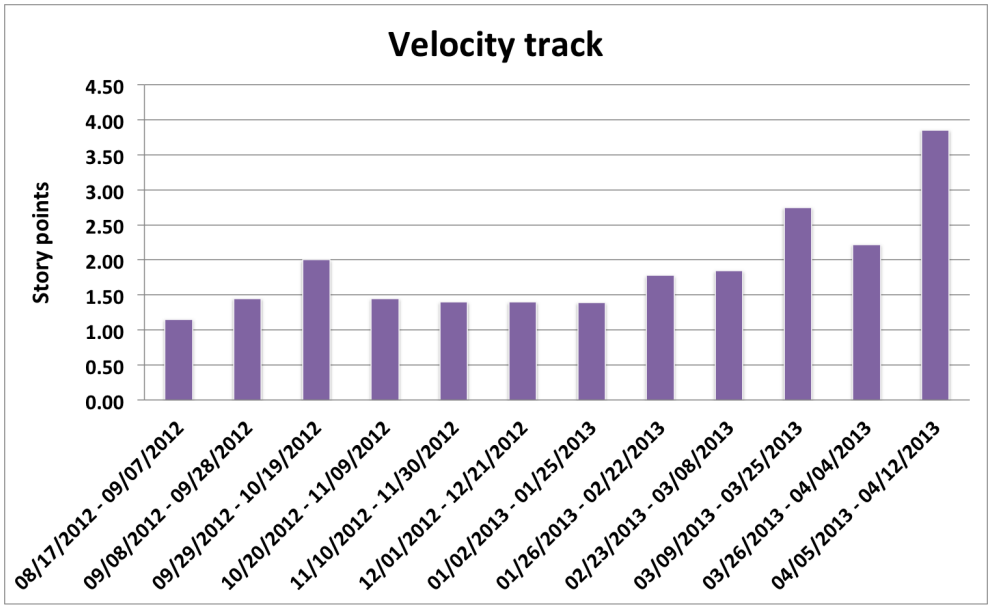


Figure 7.2: Thesis velocity track from August 17, 2012 to April 12, 2013.

iteration is because the length of the iterations varied. The graph shows that there was an increase in the workload towards the end, which may be because other subjects were completed, and my ability to estimate tasks where strengthen. When calculating the average amount of story points each iteration, weekends were not subtracted from the amount of days. This means that the average completed each period were a bit higher than what is represented by the graph, if we only consider the days used to work on the thesis. This would also eliminate days that were used on other subjects as well, which would have incased the amount of story points even more.

The velocity track of a project may be valuable for investors as it shows that the development process is still going, even though they have not seen any changes for a while. Sometimes it may takes weeks, or even months before investors see the product that is being developed. In these cases, it may help to see the velocity graph to see that there is ongoing work. It may also be valuable for the developers as it shows the amount of completed workload compared to the previously completed iterations, and let the developers see if they have done enough compared to earlier.

7.3 Source control

Source control is extremely important for software development projects, at least for medium- and large-size projects. When developing software, the source code is rapidly changing, and sometimes a small change that was supposed to solve a small problem can cause a system to break down since the code that was changed were in use by other modules as well. A source control system let the developers' rollback changes as it keeps track

of the code-lines that were changed for each *commit*. A commit is when developers update and/or insert new source code into the code base, often called a *repository*. All files that have been changed are combined with an optional comment. When this "package" is inserted into the repository, it will be inserted as one commit. In order to rollback the code, the commits are assigned a unique id, which is necessary in order to know where the code should be rolled back. Since the source control system keeps track of which code-lines that have been changed for every commit, it is capable of perform a rollback by reversing the content back to the desired commit-id. It is possible to reverse the complete code-base back to a given commit-id. However, if you do not want to reverse the whole code-base, most source control systems allows you to revert the content of just one file as well.

During the thesis work, a source control system called *Git* has been used. Git is a source control system developed by *Linus Torvalds*[1], and is one of the preferred source control systems for software developers[16]. Three other known version control systems that are popular among Java developers is Subversion, Mercurial, and Bazaar. System developers around the world also use these version control systems, and Subversion is the most used one apart from Git. We will not go into details about Mercurial and Bazaar, since they are not of interest for this thesis.

However, an important difference between Git and Subversion is that Git supports *local commits*. Local commits are commits that is committed to a temporarily repository at the local machine. There may be performed multiple commits to the local repository, which may be pushed to the main repository later on. This allows developers to work offline, even though offline development is not normal these days. If a developer create multiple local commits, and decides to push the commits to the main repository after others have made multiple commits, *conflicts* may occur. A conflict occurs if the same code-lines have been modified and committed by two or more developers. Git matches the new line, the old line, and the line that exists in the repository in order to determine if there is a conflict or not. If the old line and the line that exists in the repository are identical, the commit is completed without problems. If all three lines are identical, the line is not committed since there are no changes made. However if all three lines are different, someone else committed a change to the same line in-between the last retrieved source code and the commit. This results in a conflict, which will often require manual supervision in order to be solved.

The source code for Hecuba is available at my page on GitHub: https://github.com/baakind[17]. GitHub is a website that let developers manage their Git-repositories online. Public repositories do not cost anything, but they are available for everyone to see and *fork*. Forking means that a repository is copied, and the development progress of the fork does not follow the progress of the main project anymore. GitHub also let developers create private repositories that are available for invited developers only. These repositories are meant for companies and others

[1]Linus Torvalds is a Finnish American software engineer, which is most known as the creator of the original Linux kernel. He also created the version control system Git[25].

who would like to have an online Git-repository available for their developers, without having to invest in servers, and having someone to maintain them. GitHub has another nice feature, which is the social part. They have made coding more social by letting users comment on commits, code, follow projects etc.

7.4 Summary

The goal of this thesis is to implement an automatic scaler for Cassandra clusters. It is preferred that the implementation is generic as well, even though this is not required.

Kanban where used to keep track of the thesis work. The workflow where visualized for the thesis and the projects, which helped me keep control of the work, and maintain a high production rate throughout the thesis work. The work where also estimated to determine the workload of each task that had to be completed. I used story points to estimate the size of the tasks, which is a number representing the amount of work that have to be done. In order to estimate as good as possible, the Fibonacci sequence was used.

Git where used as a source control system, which prevented us from loosing valuable data. If changes were made that should not have been made, we could easily rollback the source to an earlier commit, which would return an earlier version of the data.

Chapter 8

Failed attempts

There were two failed attempts to implementing the autoscaler before I came up with the final solution. This chapter describes these two attempts, and why they failed and did not satisfy the goals of this thesis.

8.1 Include Hecuba into Cassandra's source code

The first attempt was to include the autoscaler directly into the Cassandra implementation as a separate single-threaded stage. The Cassandra configuration file was extended with thresholds and breach timers that were read into memory by Cassandra's already existing implementation for reading the *cassandra.yaml* file. It was supposed to share information among the other nodes through the gossip protocol as Cassandra already does, to keep track of whenever to scale down or not. Hecuba would have been decentralized since there would not be a master collecting messages and deciding which node to scale up or down.

This implementation were supposed to follow the same set of rules as Cassandra, where there are no single point of failure, and any node is likely to initiate the downscaling. In order for the implementation to successfully work, the gossip protocol had to be extended to share information whenever a node was scaling up or down. It also had to share a list of nodes that already were scaled down, and available for bootstrapping. Hecuba will not extend the gossip protocol, as it is one of the most important parts of the Cassandra implementation. Therefore, this implementation-attempt was stopped.

Changing the backbone of an implementation that relies on performance and speed where the source code is written to increase performance rather than being easy to understand is not a good idea, at least if it is not done properly. I did not want to change the backbone of Cassandra, which is currently working as intended. If I was going to change the Cassandra source code, I would have needed a lot more time in order to make sure the performance of my implementation does not lower the overall performance of Cassandra. This idea was dropped after a few weeks, when I understood that I needed more time than what I had available, in order to successfully implement the automatic scaler into Cassandra's source code.

Implementing the autoscaler directly into Cassandra's source code could have slowed down the Cassandra cluster, as it would have to send a message to all nodes in the cluster saying the node is about to be scaled down, and wait for acknowledges from all nodes before it initiate the downscaling. The node will have to wait in order to prevent downscaling of all nodes in the cluster, leaving no nodes available and the cluster would be completely shut down. E.g., There are two nodes left in the cluster. They decide to scale down at the same time, there has to be a mechanism that prevents both from scaling down. Therefore, a timestamp would have to be added to every message sent, and the node that initiated the scaling first will be allowed to proceed.

It would also require a forked version of the Cassandra implementation, and therefore it will not be possible to deploy the autoscaler onto an already running cluster. Each new Cassandra release would have to be modified with the autoscaler code, and if there are changes made to the Cassandra source code that affected the autoscaler implementation in any way, it will be necessary to rewrite the implementation.

8.2 Implemented as an extension to existing Java-projects

The second attempt became the foundation for the current solution. The implementation was separated from the Cassandra source code to prevent it from being dependent on a specific Cassandra version. However, the implementation was still Cassandra specific as it used *Hector* for handling the cluster, nodes and the communication between the nodes and the implementation. All communication were initiated from the master-implementation, and the master would ask its agents every nth second for their current status. This would potentially generate a lot of unnecessary network traffic, and eventually lower the performance or even block connections made to the cluster.

The implementation was included into an already existing project via a *Spring-bean*. Spring is a java application framework that let developers focus on the application-specific code rather than all the different environments, services, data access etc. that are part of the application[35]. It serves as a foundation for developing applications, and there currently exists a large amount of extensions for almost anything. A Spring-bean is an extended java-object, containing metadata, dependencies, etc[36].

Hecuba would be launched as a separate thread, running alongside the already existing application. The easiest way was to configure it as mentioned above, but it was also possible to initiate the autoscaler through its constructor from any java implementation as long as the autoscaler was available on the class path.

A lot of time was used to develop an implementation that communicated and were able to share messages between the master and its agents. However, it made more sense to exclude everything that were Cassandra-

specific, and everything that forced the implementation to be included into a specific type of projects or deployed to a specific platform.

8.3 Summary

In the beginning of the thesis work, I had two failed attempts at implementing the scaler.

The first attempt was to implement it directly into the Cassandra source code. The autoscaler did startup within its own thread, and were assigned a custom *stage*. However, after some time the implementation were moved out of the source code and into a separate project to prevent the autoscaler from being bound to a specific Cassandra version. Each time a new version is released, which happen quite often, the autoscale source have to be implemented all over again. The Cassandra- specific code used could be moved or deprecated, which would lead to Hecuba not working properly. It also had to be implemented into the gossip protocol, which is a well-known and efficient protocol in the heart of Cassandra. To change the gossip protocol too much could lower the overall performance of the cluster. Too many changes had to be made in order for the autoscaler to work, and therefore the implementation were moved to a separate project.

The second attempt was closer to the final solution, although all Cassandra code and communication came from the master, not the local agent. The master sent messages to all nodes every nth second, asking for the current status of the CPU-, disk-, and memory-usage. This lead to a lot of unnecessary network traffic since most of the data sent were unimportant.

Chapter 9

Hecuba design

9.1 Introduction

Currently there is no automatic up-, and downscaling implementation available for Cassandra. Even though Netflix's Priam is able to double the size of the cluster, it is not a complete automatic scaling implementation, as it does not support downscaling of nodes, which Hecuba does. Priam is also limited to Amazon Web Services, while Hecuba may be deployed to all major cloud services. Both Priam and Hecuba scales up whenever the current resource-usage breaches a certain threshold over a predefined time-period. Netflix excluded downscaling as they are afraid of scaling down too fast or too much, which may overwhelm the cluster and reduce the performance.

The current version of Hecuba does not support smart downscaling by trying to initiate downscaling at low-peak hours were the load is as low as possible, and scale down as slow as possible to monitor the health of the cluster correctly, and stop downscaling if it seems like there will be too few nodes in the cluster. Hecuba does not support downscaling where the scaling occurs during low-peak hours and when the load is at a minimum. It does not monitor the cluster health either, which should be monitored closely so an already initiated scale may be canceled if the performance drops, and the cluster suddenly run on fewer nodes than needed. However, Hecuba is intended to support smart downscaling in the future in order to successfully scale down an active Cassandra cluster.

Hecuba will keep the number of nodes in the Cassandra cluster as few as possible, without affecting the performance. As mentioned, Priam does not scale down, and therefore doesn't take into consideration the number of nodes currently active in the cluster.

9.2 Load balance issues

One of the biggest problems with the current autoscale implementation is to keep the *ring* balanced. The ring is the distribution of tokens between the active nodes in a Cassandra cluster. When a ring is balanced, the nodes in the cluster consist of *almost* the same amount of data, while nodes in

Node nr	From	To	Load
1	0000	3FFF	25%
2	4000	7FFF	25%
3	8000	BFFF	25%
4	C000	FFFF	25%

Table 9.1: Perfectly balanced 4 node cluster.

Node nr	From	To	Load
1	0000	1FFF	12.5%
2	2000	3FFF	12.5%
3	4000	5FFF	12.5%
4	6000	7FFF	12.5%
5	8000	9FFF	12.5%
6	A000	BFFF	12.5%
7	C000	DFFF	12.5%
8	E000	FFFF	12.5%

Table 9.2: Perfectly balanced 8 node cluster.

Node nr	From	To	Load
1	0000	3FFF	50%
2	4000	7FFF	23%
3	8000	BFFF	22%
4	C000	FFFF	5%

Table 9.3: Unbalanced 4 node cluster.

a perfectly balanced cluster consist of the *exact* same amount of data. An example of a perfectly balanced 4-node cluster is seen in table 9.1 on the preceding page.

Balancing the ring will always be an issue when it comes to automating the scaling process, unless the load percentage of each node is taken into consideration when applying new nodes to the cluster. However, the cluster may end up being unbalanced anyhow, as new data inserted into the cluster may be inserted into new locations, and splitting *token-ranges* will most likely not result in a perfectly balanced cluster. The tokens have to be recalculated by an external tool, and each node needs to be assigned a new token in order to perfectly balance a cluster based on the current data. The cluster will become unbalanced as soon as data is inserted or removed.

9.2.1 Token range

Token range is the range of keys a node is responsible for. A cluster may have an even distribution of tokens, but the cluster may still be unbalanced, as data may be located at *hotspots* around the cluster. A hotspot is a location of the ring where there is a greater density of data than usual.

Priam tries to solve the load balance problem by doubling the size of the cluster, were each new node pairs with an existing one[1]. When a node pairs up with an existing one, it will split the existing node's token range in half, and hopefully receive 50% of the data. E.g., if Priam were supposed to scale up a cluster of nodes seen in table 9.1 on the facing page, it would hopefully end up as seen in table 9.2 on the preceding page. Priam keeps streaming data between nodes as isolated as possible to keep network traffic down. Priam also tries to decrease the amount of time used to scale up by pairing up nodes instead of bringing nodes down and back up one by one at different locations.

It is important to know that in the example described above, the tables shows a ring which is perfectly balanced as the data is always equally distributed, and the token range is equally distributed. This will almost never happen as keys are hashed and end up as a random value. Therefore, it is hard to determine ahead of an insertion were data will be located. As a result of this, the amount of data within each token range will vary if the token ranges are evenly distributed. A token range should be calculated based on an even distribution of data across all nodes in the cluster, which most likely will result in different length of each nodes token range.

A more realistic example is shown in table 9.3 on the facing page. It is the same token range distribution as seen in table 9.1 on the preceding page, although the load balance is different. 50% of the data has been distributed to the first node, while node 2 and 3 holds most of the remaining data. Node 4 only received 5% of the data in the cluster. This is a unbalanced cluster, and the tokens should be recalculated in order achieve a balanced cluster which is as close to the cluster seen in table 9.1 on the facing page as possible.

9.2.2 Token-generation

Token generation is a crucial part of the token distribution. The cluster will most likely be unbalanced if not a complete rebalance is performed on the cluster. However, Hecuba tries to find hotspots instead of focusing on the overall load balance. Since Hecuba focuses on hotspot, the token generation will be very simple. If a hotspot is found, the token range of the responsible node is split in half were 50% is assigned to the new node, while the remaining 50% is kept by the existing node. By splitting token ranges this way, most hotspot problems should be solved. Even though the hotspot will end up at either of the token ranges, the opposite range should contain at least some load, resulting in a lowered load at the node where the hotspot occurred.

9.2.3 Load balance differences between Hecuba and Priam

Unlike Priam, Hecuba append and remove one node at a time, which may lead to an extremely bad token range distribution, and eventually end up with a unbalanced cluster. Re-balancing the ring is very costly, as it will require a lot of data to be transferred between nodes across the network. The re-balance process should never be initiated during peak-hours, as it will dramatically lower the performance of the cluster. The performance will decrease as the amount of data increases.

Hecuba will not try to keep the ring balanced, as it focuses on finding and removing hotspots as they occur. Hecuba will at the same time try to determine if the cluster consists of more nodes than needed. If it does, Hecuba will scale down nodes until it is satisfied.

9.3 Communication

The communication between the Autoscale-master and its agents happens *asynchronously*. Instead of waiting for a response the application continues, and creates a separate thread responsible for listening for incoming messages. Both the master and the agents will fork out a *"listening-thread"* to prevent other operations from waiting for one single response. Although, if a message is received which result in changes to the application, the thread listening will initiate the changes to all threads, even if they are currently executing an operation.

The master implementation will initialize the *daemon* on a master server. Usually this will be the same server as the application currently using the Cassandra cluster. The master daemon's thread will be listening for incoming messages from the agents. The master daemon will send messages to the agents through port A, and receive incoming messages at port B. These ports are currently static, and may not be changed. However, they could easily be moved to a custom configuration-file in order for developers to change the in- and outgoing ports to better fulfill their needs in case these ports collide with other applications.

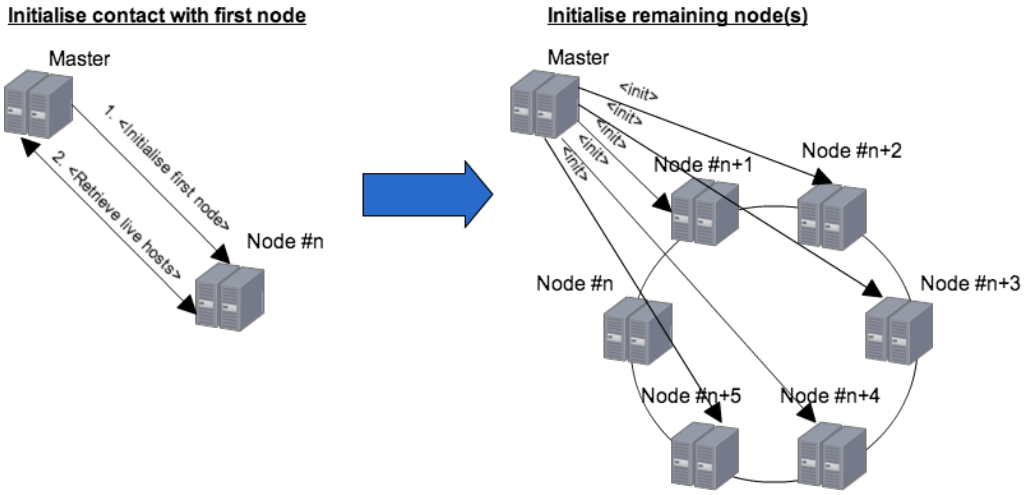


Figure 9.1: Initialization of the cluster.

The master sends an initialization-message to the initial agent first, in order to start the communication and be able to retrieve a complete list of nodes in the cluster. When the first agent is started, the master may retrieve the list, and start initializing the rest of the agents in the cluster. The master sends out messages containing default configuration values for the agents to use. This happens every time a new node is added to the cluster in the current implementation. Since the master sends out configuration attributes even when the default constructor is used, the attributes pre-loaded by the agent from its local configuration-file will be overwritten. To prevent this from happening, the master should be rewritten to prevent configuration attributes to be added when the default constructor have been used. The agents are set up to receive incoming connections at port **A**, and send breach messages to their master through port **B**; the opposite of the master.

Figure 9.1 shows how the initial phase of the implementation works. Initially, the master sends an initialization-message to the provided agent as mentioned above. The agent starts up, and the master has the possibility to ask for a complete list of active nodes. The master should know about all the nodes in the cluster when the node list is received. It may happen that the initial node does not know about all the nodes in the cluster. E.g., the initial node just joined the cluster, and therefore has not been able to gather information about all nodes yet. Although it is possible that the master may not receive a complete list of nodes, it will in most cases be complete. The master should be extended to ask for an updated list of currently active nodes from a random node in its node list every nth run, in case some nodes were not returned at the first try, or if the first try failed.

9.4 Flow

The application flow is very straight forward, but it requires Cassandra to be up and running in order to work properly. The flow described below is the *"happy case"*, whereas it often deviates because of network failures, internal problems at a node, messages not received when expected etc. The flow is just an overview of how the application works from initialization until it is fully running.

1. The nodes that are part of the cluster startup their Cassandra instances, and afterwards initializes the autoscale agent that will be waiting for messages from the master.
2. The master is initiated at the master server, usually the same server as the application using the Cassandra cluster. It will initially send out a startup message to the agent node provided upon startup.
3. The master asks the same agent for a list of currently active nodes in the cluster.
4. As the list of currently available nodes is received, the master will append everyone to a local list, except the one already known. Afterwards, the master will initiate every single node's agent, which will start monitoring its local node's resources.
5. Every node will keep monitoring its local resources, and if a breach occurs over a certain time, the node will send a message to the master containing information about which breach occurred, the breach value and which node it occurred on.
6. At a regular basis, the master will collect the received breach messages into a *batch*, and perform calculation upon them. The master will rank different breach messages according to how important they are, and if they represents a scale up or down.
7. When the calculation is complete, the master will perform scaling upon no more than **one** node. If the final score[1] of all nodes is zero, no scaling will occur. If at least one node's score is either positive or negative, a scale will occur. The node that scored highest, regardless of whether the score is positive or negative, will be scaled up or down.

9.5 Summary

Hecuba is designed to automatically scale Cassandra clusters. However, there are a few minor issues that should be fixed before the design is complete, and may be used for an implementation that will be deployed

[1]The scaling-algorithm implemented into our implementation summarize the values assigned to the messages received from the node. The result of this calculation is called the node's score (or weight).

to a real cluster. The cluster may easily become unbalanced due to the implementation removing, and inserting a lot of nodes, while it does not consider the skewness of the cluster. The design should be extended to handle token-range sharing that split the amount of data within the token-range in half instead of the token-range values.

The communication between the master and its agents happens asynchronously, which is necessary in order to let the master communicate with multiple nodes without having to wait for potential messages from the agents, depending on their current status. Initially the master will contact one of the nodes through its agent, which have to be provided upon startup. This node will bootstrap, and start monitoring. The current design does not say that this node should return a list of currently active nodes as response to the initialization-message, however it should be change to provide this list in order to be fully automatic. If not, the master will have to ask the agent again for this list, which have to be manually programmed or executed afterwards. The master will also have to be redesigned to retrieve this list from the response, and initiate contact with the rest of the nodes.

The flow of the application is very straightforward. The master is started, and contact with the initial agent is set up. The agent then bootstrap, start monitoring, and send a reply back to the master. The master will have to ask for a list of currently active nodes, which is returned as a respond from the agent. The master will initiate all the nodes that are not currently initiated, and listen for incoming messages from the agents. The master will also collect all received messages at a set interval, and perform calculations upon them. A message is sent from the master to the respective node, if the calculation resulted in the node being scaled up or down.

Chapter 10

Hecuba implementation

10.1 Introduction

The implementation consists of two separate Java applications: a master-application, and one or more agent-applications. The master is responsible for collecting statuses from each node in the cluster through the agent, and initiate up- or downscaling if necessary. The master may be included into an already running web-application by downloading and building the project, or as a standalone application.

In order to retrieve data from the systems CPU-, memory- and disk usage, a third party framework named *SigarAPI* is used[22]. SigarAPI creates an abstraction layer, and interfaces the commands used to retrieve system statistics, since they are specific to each OS. There is not a good idea to include OS-specific scripts within a Java application, as it should not be OS dependent.

There are some known bugs and shortcomings in Hecuba, and the implementation is not ready to be deployed outside a test environment yet. There were not enough time to fix all the bugs that were found, and further develop the implementation in order to satisfy the requirements for a fully working autoscale implementation. Hecuba is open sourced under the *Apache License*[1], and is currently available at:

- Autoscale: https://github.com/baakind/autoscale
- Autoscale-agent: https://github.com/baakind/autoscale-agent
- Autoscale-common: https://github.com/baakind/autoscale-common

10.2 Code separation

The implementation is divided into three separate projects: *Autoscale*, *Autoscale-agent* and *Autoscale-common*.

[1]The Apache License is an open source license that let anyone use or distribute the software as they like, without having to pay. The only requirement is that the license and the NOTICE document are distributed together with the software if present. The license may be found at http://www.apache.org/licenses/LICENSE-2.0

- **Autoscale** - The master implementation. It is responsible for collecting messages sent from the agent(s), and determines if the cluster needs to be scaled up or down. The autoscale project depends on the *Autoscale-common* project for the communication, and message objects, that is shared between the master and agent implementation.
- **Autoscale-agent** - The agent implementation. One agent-instance will be running alongside each node in the cluster. The agent will monitor the local node's resource usage, and send messages back to the master if a breach occurs over a predefined timespan. The agent project depends on the *Autoscale-common* project for the communication and messages objects.
- **Autoscale-common** - Contains the objects that the *Autoscale* and *Autoscale-agent* uses. Currently the message-objects and the communicator is part of this project.

10.2.1 Autoscale

Hecuba's master-implementation. The autoscale-implementation is responsible for collecting messages sent by the agent(s). At a certain time interval, the master will iterate all messages that are currently collected, and deter- mine if the cluster should remove or append nodes. There are currently four different types of breach messages[2] that may be sent from the agent(s):

- *max_memory_usage* -The memory-usage of the node has exceeded a certain threshold for *n* seconds, defined by the *thresholdBreachLimit* attribute.
- *min_memory_usage* - The memory-usage of the node has been less than a certain threshold for *n* seconds, defined by the *thresholdBreach-Limit* attribute.
- *max_disk_usage* - The disk-usage of the node has exceeded a certain threshold for *n* seconds, defined by the *thresholdBreachLimit* attribute.
- *min_disk_usage* - The disk-usage of the node has been less than a certain threshold for *n* seconds, defined by the *thresholdBreachLimit* attribute.

The default scaler class combines each breach-type with a value, saying if the node should be scaled up or down. The value also represents the importance of the breach-type. A breach-type indicates that the node needs to be scaled up if it is assigned a positive value, while it indicates that the node needs to be scaled down if it is assigned with a negative value. A higher positive value has higher importance than a lower positive value for which node to scale up, while a lower negative value has higher

[2]A breach message is a message sent by an agent, telling the master a certain threshold has been broken over a given time period.

importance for which node to scale down. The values may be found in the `SimpleCassandraScaler` class, which may easily be replaced by a custom implementation.

Apply to your own project

The autoscale master-implementation is a standalone implementation that may be applied to a project in different ways. It may be implemented as a spring bean if the project uses the spring framework. The jar may either be put directly into the project's class path, or it may be built locally to append the master implementation to the local maven repository if the project uses maven. When the master implementation is located in the local maven repository, it may be included as a dependency for the project that is going to implement Hecuba, and initiated by creating a new instance of the `Autoscale`-class.

The implementation may be initialized by the default constructor, which does not require any parameters, or by the constructor that require the most important configuration-attributes as parameters. The parameters define the thresholds and number of seconds a breach occurs before a message is sent from an agent to the master. If it is initiated without parameters, the default parameters will be used both for the master and the agent(s). By initiating the implementation through the default constructor, each agent may be initiated with a custom set of thresholds and timers, as they may be retrieved from the local configuration file instead of retrieving them from the master initially. In most cases the nodes should be configured with the same parameters, as they should be running on the same type of hardware. However, it may happen that one or more nodes are running on different hardware, and therefore should have different thresholds. A detailed description of the parameters used for the constructor is described in table 10.1 on page 78.

The Autoscale-master flow

The master will be initiated through e.g., a spring-bean or directly as a java object within an application. The implementation should be initiated from the *Autoscale* class. The implementation will initiate *AutoscaleDaemon* through a thread, running every second. The daemon is the actual master implementation, while the Autoscale-class only works as a startup container enabling the daemon to be initiated at a scheduled interval. A visual representation of the master implementation is seen in figure 10.1 on the next page.

After the master implementation has been initiated, it will initiate contact with the `initHostname:initPort` (described in table 10.1 on page 78) node. To successfully startup the application, the initial node should already have a running instance of the agent implementation in order to receive and respond to messages from the master.

At the same time, the master will initiate two separate threads to run simultaneously: the *AgentListener* and the *SimpleCassandraScaler*. The

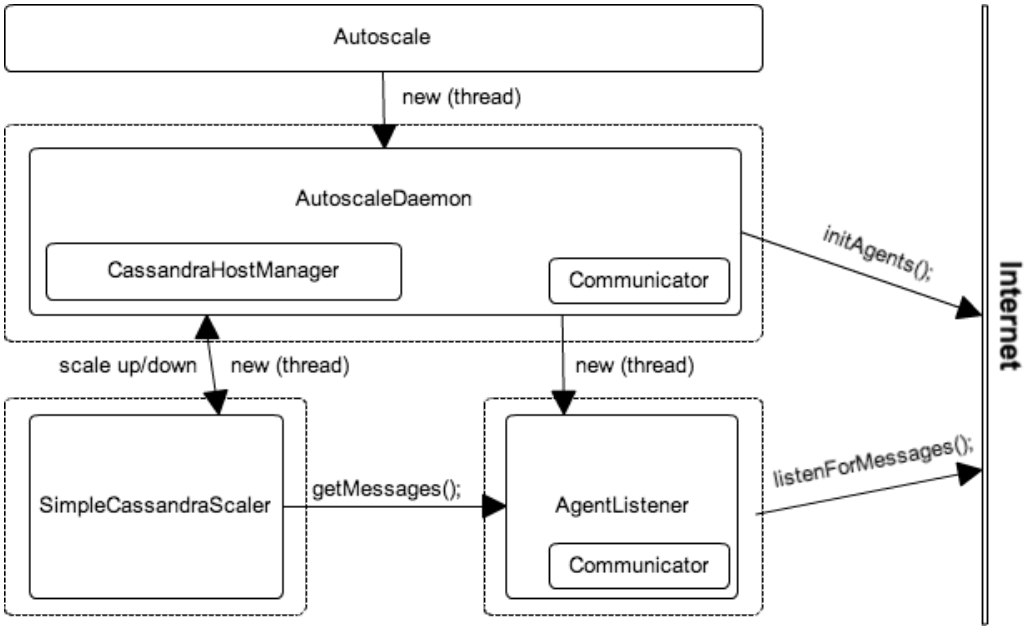


Figure 10.1: Visual representation of the Autoscale-master implementation.

AgentListener will be scheduled to run each second in order to catch all received messages. The listener will setup a socket, listening for incoming messages from the agents. Messages received will be added to a local message list, and the listener will continue listening for new messages. The scaler will also be running at a predefined interval, which may be changed by the initial parameters to determine how often a scale should occur. The scaler will collect the current list of messages from the listener, and make sure its empty afterwards in order to prevent reading the same message(s) twice. The scaler will calculate whenever nodes should be scaled up or down after the messages is retrieved. The current implementation will compute *weightedScoring* for each node. Each node will be assigned a *weight* based on the messages received from the node. As mentioned, positive integers are assigned to messages that indicate a node being scaled up, while negative integers are assigned to messages that indicate a node being scaled down. The highest weight will be prioritized when deciding which node should be scaled up or down.

At this point, the master implementation should be listening for incoming messages from the agents. If the initial agent started without any problems, and the incoming and outgoing ports are open, the master may send a message to the agent asking for a list of available nodes. This is currently not a part of the implementation, and has to be initiated by the applications that implement Hecuba. However, each agent should send a node-list of the currently active nodes in the cluster as a response to the initial startup-message. The master will match the local list of active nodes with the received list. Currently, the master's list only contains the initial node, as it has no knowledge of the cluster yet. However, the list returned by the node will contain all nodes in the cluster. In most cases a Cassandra

cluster consists of more than a single node, which will result in the master sending initial startup messages to all new nodes, and at the same time appending them to its local list of active nodes. It is highly recommended that all nodes in the cluster are running the agent implementation at this time, in order for all nodes to receive the initial startup message and start monitoring and messaging.

10.2.2 Autoscale-common

Contains the common objects for both the master and the agent implementation. Both projects depend on this in order to make use of code, which is identical to both projects. It contains the *Communicator*class, which is used when setting up sockets for sending and listening for messages. The messages sent by the communicator may be either an *AgentMessage* or a *BreachMessage*. The received objects will be put *CommunictorObjectBundle*. The CommunictorObjectBundle object is just a wrapper that wraps in the message and the senders IP-address.

- **AgentMessage** - Messages sent between master and agent. If it is sent from the master to the agent, it contains instructions for the agent, while it contains a breach message if it is sent the other way. The message may also contain an optional map of attributes. The different types of instructions that may be sent are:
  - `STARTUP_NODE` - Initializes the node, startup the agent and update the local configuration's if settings provided by the attribute map. There will not be sent attributes if the master is started with the default constructor, and the agent will load its settings from the local configuration file.
  - `UPDATE` - The agent will update its current configurations with configurations found in the attribute map.
  - `STOP_AGENT` - The agent will stop monitoring and sending breach messages.
  - `START_AGENT` - The agent will start monitoring and sending breach messages.
  - `SHUTDOWN_NODE` - The running Cassandra instance is shutdown. The data will be distributed to other nodes in the cluster and the node will be decommissioned. After it has been decommissioned, all data found in the *data*, *commitlogs* and *saved_caches* would be removed, preparing the node for entering the cluster at another location. The data directories may be changed in the agent's configuration file.
  - `STATUS` - The agent will return a status depending the received `AgentStatus`. Currently the only implemented AgentStatus is *live nodes*. It returns a list of the current available nodes in the cluster. However, it may easily be extended if needed.

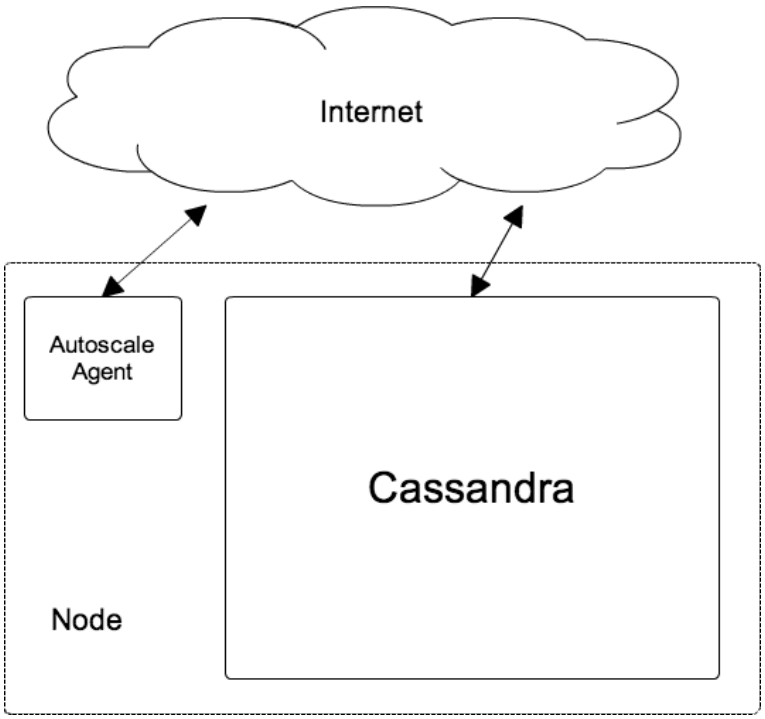


Figure 10.2: Cassandra and the Autoscale-agent running alongside each other.

  - `BREACH_MESSAGE` - Sent from the agent to the master, indicating a breach has occurred. If this type of message is received, the breach-message will be included in the attributes-map.

- **BreachMessage** - A breach message contains information about a breach that occurred at one of the nodes in the cluster. The current breaches are either *maximum-* or *minimum memory-usage*, or *maximum-* or *minimum disk-usage*. The message also contains the responsible value at the time of the message being sent. If there is a memory breach, it contains the memory-usage percentage, and if there is a disk breach, it contains the disk usages in Megabytes. The breach message will be wrapped into an agent message. The agent message's type will be set to `BREACH_MESSAGE`, while the breach message will be put into the attributes-map. Since the breach-message is put into the attributes-map, Hecuba is able to send multiple breach messages at the same time, although this is not implemented in the current release.

10.2.3 Autoscale-agent

The agent implementation is acting as a *slave* to the master implementation. It should be running on all nodes in the Cassandra cluster to monitor it is current resource usage. The agent will run alongside the Cassandra implementation as seen in figure 10.2.

The agent is responsible for monitoring the local node's memory and disk usage. If the values falls below, or exceeds a set of thresholds for a certain time, a message is sent to the master. The message contains the agent's IP address, type of breach, and the readout value as the message was sent. When a message is sent, the timer is reset, and the agent continues to monitor and report. It will repeat the process as long as a breach occurs, or until a message is received from the master asking the agent to either stop monitoring, or to shut down the node.

How to implement the agent

To adopt the agent implementation and initialize it on a running Cassandra cluster, download the source code from the links found at page 65. An archive file is located in the *release* folder, which contains everything needed to successfully deploy the agent implementation. However, it is possible to make changes to the source code by downloading the agent- and common projects. Make sure to successfully generate the target-folder and having a copy of the jar files for the common project within the local maven repository before the agent project is built. If the common project is not found in the local maven repository when building the agent, it will fail. The agent depends on the common implementation, which are not found in the official maven repository at this time.

If the archived file were downloaded, it may be unpacked and the content moved to a desired location at the respective node. The agent includes a configuration file found at *conf/autoscale-agent.yaml*. The configuration file should be changed before initializing the agent, or else the implementation will not work properly. A list of the most important attributes is shown in table 10.2 on page 79.

After changing the attributes to match the node where it has been deployed, the agent may be initialized. To initialize the agent, go to the root of the agent-folder, and execute the `bin/autoscale` script. The agent may be logging a few messages to the console, depending if the log-settings were changed in *conf/log4j.properties*.

If the agent only returns empty values only, or it failed while trying to read the current system resources, there may be something wrong with the SigarAPI files provided. All the SigarAPI modules are provided, and are located in the `lib/sigar-n.n.n` folder. Try updating the version by downloading the newest release from http://sourceforge.net/projects/sigar/files/. Unpack and copy the content into `lib/sigar-n.n.n` and replace the existing files. By default libraries for all operating systems is included. All files except the jars and libraries corresponding to the operating system where the implementation will be deployed may be removed.

The Autoscale-agent flow

The Autoscale-agent implementation is initialized by a bash-script located in the *bin* folder. The script constructs the class path, including all files found in the *lib* folder, as the agent uses them all. The script will initiate the

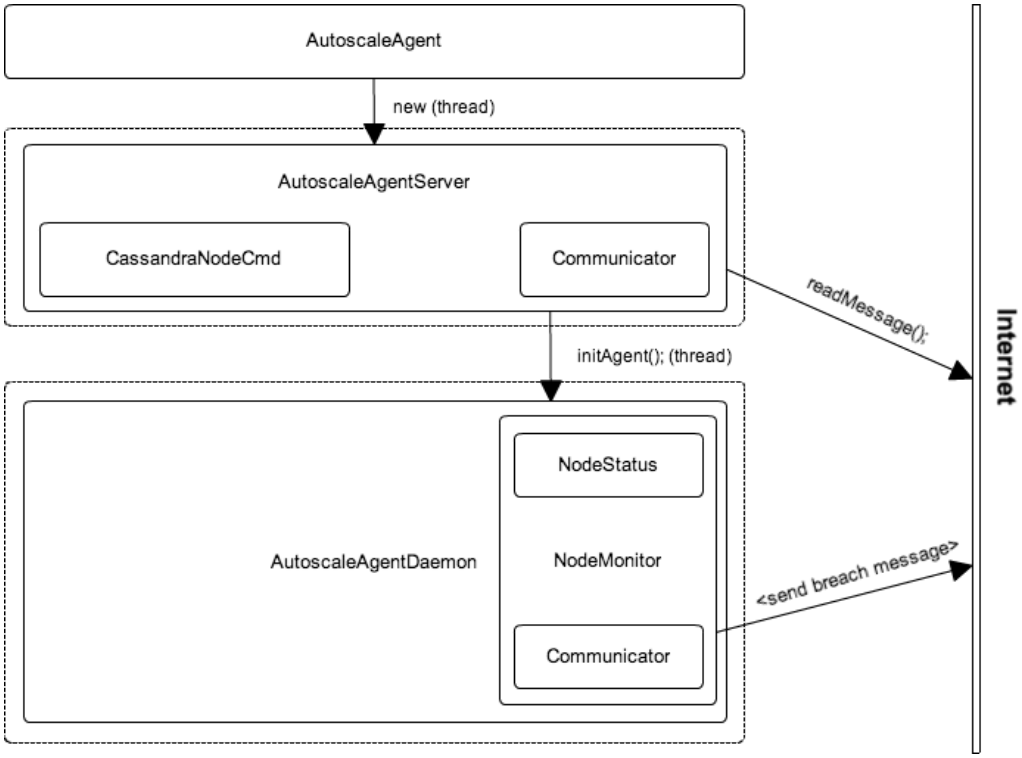


Figure 10.3: Visual representation of the Autoscale-agent implementation.

implementation through the `AutoscaleAgent` class. AutoscaleAgent works as a wrapper to startup the actual autoscale application. It will retrieve the interval timer for the agent from the configurations, and initiate the agent-server as a separate thread. The interval-timer that determines how often the agent-server will be running cannot be set externally since it is loaded before any messages or configuration-files are read. A visualization of the agent implementation is seen in figure 10.3.

The `AutoscaleAgentServer` class will load the configuration file into memory and update its local attributes. The AgentServer will initiate the communicator and start listening for messages. Nothing happens before a message is received. After the initial `START_AGENT` message is received, which is sent from the master, the agent will initialize its daemon, the `AutoscaleAgentDaemon`, in a separate thread at the interval set by the configuration attribute `interval_timer`. The AutoscaleAgentDaemon will monitor the disk and memory usage for the local node during each iteration. If a breach occurs, the counter representing the actual breach is increased by the interval-time variable.

Whenever the counter exceeds the threshold that is defined, a breach message is sent to the master. The message contains what type of breach that occurred, and the readout value corresponding to the breach type when the message was sent. Each time a message is sent, the respective timer is reset, and the monitoring continues.

10.3 Tools and frameworks

10.3.1 Maven

Maven is a popular project management tool used by software developers[27]. Maven is based on a project object model (POM) which is used to man- age and structure the project(s). Each project that uses Maven needs to have a `pom.xml` located in the root directory. The POM-file contains the *"recipe"* maven uses when performing actions upon the project. There is an endless number of possible actions to be performed, as there are extensions available for almost everything. The POM-file must contain information about the project's name and version. It must also contain the information whether the project is a jar (java archive) or a war (web archive). It may contain one or more dependencies to java libraries that will be included in the class path, or links to local sub-projects if any. The POM-file may also contain deployment information, which automates the deployment to a server if desired. There is an endless list of possibilities that will not be described in this thesis, as it is outside the scope.

10.3.2 SigarAPI

SigarAPI is an acronym for *System Information Gatherer And Reporter Application Programming Interface*[22]. Sigar is developed as a cross platform solution that works on most known platforms that exist today. The API provides functionality for gathering detailed information about system resources like the CPU's, memory, disk and network usage.

Sigar interfaces the layer between the implementation and the system specific commands by providing an individual set of instructions written in C for each operating system, as each OS have their own way of representing its resource data. There are implementations with bindings to the instruction sets available in Java, Perl and C#. It is released under the Apache 2.0 license, and the complete source code is available online.

10.4 Scaling

10.4.1 Default scaler

`SimpleCassandraScaler` is the default scaler that comes with Hecuba. The implementation contains a simplified scaling algorithm that collects the current batch of received breach messages, and performs a simple calculation. During each iteration, the scaler will retrieve the current collection of breach messages from the listener. After the list is retrieved, the scaler will empty the list in order to prevent retrieving duplicates during the next iteration. The frequency of the iterations should be set a few times higher than the breach-time limit for the agents. There will not be any messages to perform calculations upon if the agents have not yet sent them. If the iteration-frequency is set to e.g., three times the breach-time-limit for the agents, the scaling-algorithm may at most have three messages from

each node to perform calculations upon. I would recommend the iteration-timer to be at least three times the breach-time-limit, but preferably higher in order to achieve a more accurate calculation.

When the scaler have copied the collection of breach messages, it will start generating *host weights* for each node. The host weight is a score assigned to each node, indicating if the node should be ignored or scaled either up or down. The host weight is calculated by iterating the breach message collection. For each message, the *priority* of the breach message is retrieved. The priority is a predefined integer assigned to each message type that indicates either a scale up or down. If the node already has a value assigned, the new value is added together with the existing, and the record is updated with the new value. If the node does not have a value yet, the value is assigned to the node, and the node is appended to the list of existing nodes. When the calculation is complete, a list of nodes with a score representing either a scale up or down is returned.

Afterwards, the scaler selects one of the nodes, which should be scaled up or down, if there is any. If the weight for all nodes is zero, nothing will happen, although that will probably not happen, as nodes rarely send a combination of scale up or down messages at the same time. The scaler will split up the nodes into two lists, one representing nodes which has a positive score, while the other list contain nodes which has a negative score. The scaler will extract the highest positive score, which represents the node that should be scaled up, and the lowest score, which represents the node to be scaled down. The lowest score will be converted it into a positive integer so they can be compared. If the node that are going to be scaled up have a higher weight-score than, or equal to the negative score of the node with the lowest score, the node that should be scaled up is prioritized. If the scale down score is higher than the scale up score, and the scale down list contains at least one node, the node with the lowest score will be returned. If there were a node selected to be either scaled up or down, the scaler will initiate the scaling, and update its internal list of active nodes.

10.4.2 The simplicity of the default scaler

The default scaler is very simple. It does not consider the value returned by the nodes, just the type of message. E.g., a node that has less than $100MB$ of disk space left be prioritized before a node that has $900MB$ left. This is a know problem, but it is not a part of the calculation, although the data is available through the breach message, and may be used by a custom scaler if necessary.

The scaler does not prevent scaling to occur during peak hours or holidays. The scaler could have implemented functionality that prevented scaling to occur at certain timespans, or if the network traffic where too high. Today the agent does not monitor the network traffic, even though it may be extended with such functionality in the future.

10.4.3 Implement a custom scaling algorithm

It is easy to develop and implement a customized scaling algorithm. The class has to implement the `Scaler` interface, which will require you to implement all the required methods. The scaler is created within the `AutoscaleDaemon` class, and not included through either an input parameter or read from a configuration file. The custom scaler has to replace the SimpleCassandraScaler within the AutoscaleDaemon class in order to use the custom scaler.

10.5 Scripts

Some parts of the code are OS-specific, and may not be executed as java code. It is possible to interface it in the future, but currently the implementation supports changing the startup- and shutdown command. The startup command is used to bootstrap the Cassandra instance and join the ring. The command is executed through the `Runtime` class provided by java, which enables execution of commands directly to the OS. The shutdown command does the opposite, as it decommission and shutdown the node. Since the Cassandra instance is not a part of the Hecuba implementation, there does not exist a direct link between them, even though both are Java programs. Java executes each single instance in its own *sandbox environment*. A sandbox environment prevents two widely different java programs to interact directly. Since there is no easy way for Hecuba to communicate directly with the running Cassandra instance through a Java-interface, and still be generic, Hecuba executes the commands directly to the OS. The OS has knowledge about all processes currently running, and may therefore terminate or setup the Cassandra process accordingly.

The startup script for the agent is written as a bash script. There is currently no solution available for Windows machines. The script initialize the class path by appending all jars from the *lib* folder, and all files found in the *lib/sigar-n.n.n* folder. The files found in the library folders are used by the application as third party libraries for retrieving system resource usage, logging, communicate with Cassandra etc. To be able to use these libraries, they have to be appended to the class path so the java program is able to locate them. The script also appends the path to the logging properties-file, which set the logging- level of the application, and the location of the output file(s). The output file(s) may be one file, separate file for each log level, console, etc. After everything is set up and prepared, the `AutoscaleAgent` class is executed.

10.6 Limitations

There are a few limitations as the implementation is not completely done, and some priorities had to be made. The application has not focused on

performance or smart solutions as the main focus were to create a working *"alpha version"* of an automatic scaler.

At this time, Hecuba only works for Cassandra. Hecuba was originally not going to be a generic implementation that worked for all major databases, as well as other kind of distributed software, which was in need of an autoscaler implementation. Although as the implementation were developed, it was possible to generalize a lot of the functionality, which led to an implementation that may be generalized and separated from the Cassandra-specific code.

The implementation is currently built for Linux. The java code will work anywhere, but the scripts to e.g., initialize the agent, and shutdown or startup the Cassandra instance are bash-scripts, which is written specifically for Linux. The application should be OS independent, and therefore the scripts should be removed or at least generated for every major OS.

Hecuba is not able to scale infinitely as Priam may. Priam doubles the size of the cluster on Amazon Web Services (AWS) as it may execute commands towards Amazon's API to setup new Cassandra instances which will be started, and inserted into the cluster. Hecuba does not have this opportunity, as it is not built specifically for AWS. Hecuba should successfully be deployed onto AWS without problems, although it has not yet been tested and verified. To solve the *"infinite"* scale-problem, the database administrator may initialize n number of nodes into the cluster. After some time, Hecuba will stabilize the cluster by scaling down nodes that is not needed. For Hecuba to be able to scale down, it is important to initialize the agent implementation alongside the Cassandra instance on each node. By firing up enough nodes, Hecuba will keep a list of the nodes which have been shut down and available to be put into the cluster whenever hotspots occurs or nodes are overwhelmed with traffic or data.

10.7 Summary

Hecuba is developed to be as generic as possible. Most of the Cassandra-specific code is interfaces, even though there was not enough time to completely abstract everything. The main goal of the thesis was to implement a successful scaler for a Cassandra cluster, however it is desirable to create a generic implementation that successfully work for all major database solutions today.

The code is separated into three projects: *Autoscale*, *Autoscale-agent* and *Autoscale-common*. The autoscale project holds the master implementation, which is responsible for initializing all the agents, collect breach messages and scale up or down nodes if needed. The autoscale-agent project holds the agent implementation, which is responsible for monitoring the resources for the local node using SigarAPI. SigarAPI interfaces operating system specific commands for retrieving CPU-, disk-, memory-usage and network traffic. The node will monitor the resources used at a given time interval and send breach messages to the master if breaches occurs over a

certain time based on pre-defined thresholds. The thresholds may be sent from the master when initializing or updating the node, or read from a configuration file located at the agent. The common project contains code that the master and the agent implementation have in common. Currently the common project contains the communication code, since this is the only code both projects have in common.

The default scaler that is implemented is relatively simple. If a scale up is initiated and there are available nodes, the scaler will startup the new node at the token range that currently have the heaviest load. The location of the node is automatically found by Cassandra, but may be overridden by implementing a custom scaler. To implement a custom scaler the scaler interface has to be implemented, and the SimpleCassandraScaler code from the AutoscaleDaemon class have to be changed to the custom scaler.

Argument	Description
`intervallTimerAgent`	Interval-timer that tells how many seconds there should be between each time the agent monitors its resources.
`intervallTimerScaler`	How many seconds between each time the scaling algorithm should collect messages. The timer should be set to at least the same as `thresholdBreachLimit` to be able to collect all messages. It is recommended that the attribute is set to at least three times higher for the algorithm to perform calculation upon more than just a few messages. Too few messages may result in incorrect calculations.
`thresholdBreachLimit`	Tells how many seconds a breach should occur before a message is sent from an agent to the master. This is used to prevent spamming messages to the master if e.g., the memory is filled up just before a compaction, and goes back to normal after a few seconds.
`minNumberOfNodes`	The master should never scale down the cluster below this number, as this is the absolute minimum number of nodes that should be up and running at any given time.
`minMemoryUsage`	Minimum allowed memory-usage as percentage for the node. If the memory-usage is below this threshold for n seconds, a message is sent to the master.
`maxMemoryUsage`	Maximum allowed memory-usage as percentage of the node. If the memory-usage is above this threshold for n seconds, a message is sent to the master.
`minUsedDiskSpace`	Minimum used disk-space in Megabytes. If below this threshold for n seconds, a message is sent to the master. The message tells the master that the node which sent the message is using less disk-space than the node is set to use, and should be scaled down if possible.
`maxUsedDiskSpace`	Maximum used disk-space in Megabytes for the node. If above this threshold for n seconds, the node is running out of available disk space. It may be fatal if nothing is done, and data is inserted into the cluster. This message should be prioritized to prevent loss of data. Another node should be inserted into the cluster at the token-range for the actual node to relieve it.
`initHostname`	The hostname or IP-address for the agent that will be used during the initial startup of the implementation. The other agents will automatically be appended if the master asks for active hosts from the agent.
`initPort`	Port-number used in combination with `initHostname` to make the initial connection..

Table 10.1: Startup arguments for the Autoscale-master implementation.

Attribute	Description
`root`	The complete path to Cassandra's root-directory.
`startup_command`	The command executed directly to the operating system to startup the Cassandra instance.
`shutdown_command`	The command executed directly to the operating system to shutdown the Cassandra instance.
`clear_directories`	Directories that will be emptied whenever the Cassandra instance is successfully shutdown: • **data**: Hold the data inserted into the cluster, which the local node is responsible for. • **commitlog**: All commits made to the node. Data is removed from the commit logs after it has been flushed to disk, and included into the local data set. • **saved_caches**: Temporarily cached data for the node
`node_address`	Used to connect to the local node-command interface.
`node_port`	Connection-port used in combination with the argument above.
`input_port`	Port where data is received from the master.
`output_port`	Port used to send data to the master.

Table 10.2: The most important configuration-attributes for the Autoscale-agent implementation.

Chapter 11

Test results

11.1 Introduction

This chapter describes the tests performed to prove that the implementation works. In order to determine the severity of the impact made by Hecuba, the tests have been performed with, and without Hecuba running. The test cases were not supposed to prove efficiency or performance, but rather demonstrate that the implementation actually works. There is a lot of work to be done before the implementation may be deployed to a live cluster, although the basics of an autoscale-implementation are complete.

The test cases recorded results using the logging script: *bin/logging*. The logs were extracted from both nodes, and converted into CSV files (comma separated values). The CSV files were used as a foundation to generate the graphs visualizing the test results. Each test (except test 3) was performed twice, one time with-, and one time without Hecuba running. This was done in order to prove that Hecuba has little or no impact on the performance of the cluster.

The network traffic was not logged during the test cases. It could have provided valuable data to be analyzed since it would have shown the amount of data transferred between the nodes while the implementation was running, compared to the transferred data while the implementation was not running.

The test results focuses on the memory-usage of the nodes in order to determine how well Hecuba performs. The CPU-, and disk-usage was also monitored, but the results were not interesting enough to be described in details. Cassandra tries to keep as much data as possible in main memory in order to increase the response time. Hecuba does not keep anything stored on disk, and runs solely from main memory. Since Cassandra and Hecuba keeps most of their data within main memory, visualizing the memory-usage will give a good understanding of how they work, and impact the performance of the nodes and the cluster.

The logs that were recorded from all tests are found in the appendices at the end of this thesis. These logs consists of the memory-, CPU-, and disk- usage that were recorded every fifth second from both nodes, during all seven tests cases.

11.1.1 Linode cluster

The tests were performed on a cluster provided by a cloud service named *Linode*[24]. Linode is a cloud service that let users rent virtual servers in the cloud. They currently let you choose from six different data centers located all over the world, which gives you the freedom and opportunity to locate the data as close to your customers as possible. The user will have full root access to the operating system, and access to the servers' statistics like network traffic, CPU usage and disk IO operations. The user may also deploy any Linux distribution they like from a pre-defined list, consisting of all the major distributions currently available. Linode does not support any other operating systems, since it is a Linux-only service.

The test cases were performed on two nodes located in New Jersey, USA and London, England. They was meant to demonstrate that the scaling occurred, and Hecuba's impact on the overall performance. As it was only two nodes running, the tests would also end up scaling down to the minimum number of nodes allowed in a cluster, which would prove if the implemented functionality for keeping a minimum number of nodes in the cluster worked. Both nodes used for all test cases were running on Ubuntu[1].

11.2 Goals and expected results

The goals of the test cases was to provide a *"proof of concept"* that the implementation actually works, and that it does not impact the node's resources in a negative way. The tests will prove that the performance is not lowered due to the automatic scaler running in the background, which will be best seen by looking at the memory usage of the nodes. The cluster holds as much data in main memory as possible, and since there are not much data available in the cluster during these tests, the memory usage will be a good representation of the node's workload, and to see if the scaler makes an impact or not. By comparing the memory-usage when the autoscaler is running, against the memory-usage when it is not running, will give information about the severity of the impact made by Hecuba. We expect Hecuba to have some impact, since it requires memory to run. It will also be responsible for initiate downscaling of nodes, although the usage should not deviate from what is normal.

The scaling should occur

Our main goal of the test cases is of course that the scaling occurs, or else the implementation will be useless. The scaling process should be triggered based on a set of attributes provided by either the constructor or a configuration file. If the attributes are retrieved from the configuration file, each node may be set up to trigger at different thresholds and different

[1]Ubuntu is one of the most popular Linux distributions available. More information may be found at their official website http://www.ubuntu.com/.

time intervals. Since the configuration attributes may be set separately for each individual node, the implementation is way more flexible than if the master set the attributes. The reason why some nodes may require different trigger-threshold than others may be because some of the nodes have e.g., larger disk- or memory-capacity than the rest. Even though it may happen that nodes in the cluster are composed of a variety of different hardware, this rarely happens for cloud services as all nodes are often composed by the same commodity hardware.

However, it may happen that some of the nodes are running other applications simultaneously as they run Cassandra. If there are nodes that run multiple applications, they may be scaled up prior to nodes which only run a Cassandra instance and the autoscaler to prevent lowering performance of the other applications running. Nodes that have other applications running at the same time will most likely be scaled up first, at least if the specifications for all nodes in the cluster is the same. They will be scaled up earlier because the maximum-memory usage threshold is reached earlier than nodes without other applications running.

Makes little or no impact to the cluster

We expect that the cluster will continue to perform as if Hecuba were not running at all, or at least continue to run without any greater impact to the performance of the cluster. Hecuba will of course have some impact to the memory- and CPU-usage of the node, but hopefully the impact from Hecuba will not affect Cassandra's performance. The main idea for Hecuba is that the implementation will be an extension to Cassandra that may be deployed to both existing- and new clusters. The users and the administrators should not notice any performance loss due to Hecuba running.

The current implementation may trigger the scaling process during peak hours, which may result in performance loss. This is an issue that should be prioritized for future deployments of Hecuba. If a scale occurs during peak hours, nodes that are currently interacting with users may be scaled down and cause unknown errors. There will also be an increase in the amount of data transferred between nodes, and all nodes in the cluster will have to adjust their token-ring information in order to direct traffic to the correct nodes after the scale occurred. All this will require processing time and bandwidth, which occasionally will affect the user experience and the overall performance.

Therefore, it is highly recommended to implement a time-window where the scaler may or may not be triggered. The implementation should also be further extended with a module that detect, and prevent scaling to occur whenever there are upcoming peak hours.

The cluster should not be inefficient due to the scaling

We expect that the cluster will not loose performance or be inefficient due to the unbalanced ring. After a certain time, the ring will end up

being unbalanced due to the removal- and insertion of nodes. Nodes will be removed from the token-ranges that have the lowest amount of data and traffic, while nodes will be inserted into the token-ranges where a *hotspot* occurs. A hotspot is a location in the ring where there is a greater density of data than elsewhere. Hopefully the cluster will not suffer due to the unbalanced removal and insertion of nodes, although this is likely to happen after a while. Hecuba does not have the functionality to detect an unbalanced cluster, although it would be a good idea to implement such functionality. Hecuba should be extended with functionality for detecting whenever a cluster have become too skew, and trigger some sort of re-balancing process that is triggered during low peak hours.

If three nodes hold an equally large amount of the total keyspace (ring) of a given cluster, they are responsible for 33.3% of the overall amount of data each. Data inserted into the cluster may or may not be equally distributed among the nodes, as the key is hashed, and therefore ends up at a random position in the cluster, based on the hashing algorithm. As the positioning of the inserted data is hard to determine ahead of the insertion due to the randomness of the generated hash, it is impossible to pre-define a token-range distribution that will keep a perfectly balanced cluster at all times. Therefore, it is recommended to always rebalance the cluster periodically in order to keep the cluster as balanced as possible.

Even though the cluster will end up being unbalanced faster when Hecuba is running than if it was not, it will hopefully not result in a noticeable performance loss, or nodes ending up being responsible for the majority of the data. As mentioned earlier, Hecuba may solve this by periodically recalculate tokens for all nodes in the cluster. This will move nodes to new locations, and data will be re-distributed among the nodes. Each time a cluster is re-balanced, the cluster will spend a lot of resources transferring data between the nodes. As the nodes will be busy reading-, exchanging- and compacting data, they will not perform as good as they should if they have to serve hundreds or even thousand of requests at the same time. If the skewness of the cluster, and the module responsible for re-balancing the cluster, leads to performance-losses which lowers the overall user experience and the response time, Hecuba would have to be re-programmed.

Hecuba may be re-programmed to work the same way as Priam does, which efficiently double the size of the cluster by pairing each new node with an existing one[32]. However, Hecuba will also have to halve the size of the cluster each time a scaled down occur, which Priam does not support. It may potentially lead to a huge performance loss, since the performance may drop below what is required.

Should give an indication of whether Hecuba is a potential success or not

The test cases should indicate if Hecuba might be a potential success or not. The test cases will be performed on a small cluster consisting of two Linux nodes, with a small amount of data inserted into the cluster. The tests will

only run for 15 minutes, as it should be enough to determine if the basic implementation works or not. The tests will only record results just before, when, and after a scale occurred, if it occurred at all. Since the tests are very simple and performed over a short period of time, they do not provide the results necessary to create a final conclusion whether the implementation is a success or not. However, the test cases will indicate if the implementation may work, and if Hecuba should be further developed.

The preferred testing environment would have been a larger cluster consisting of at least 10 - 20 nodes, with a larger amount of data available, and the tests were running longer. This would put the implementation through a more realistic test case, and would prove if the implementation works over time, and not only the moment when the scaling occurs. It would also test all the other factors that have to be tested, like the network traffic, organizing large amounts of received data, scaling while hundred of connections are made etc. The data will be shuffled around between the nodes, and would potentially be hard to retrieve. These tests would have been preferred, but it was not possible to perform such tests with the current implementation, as the implementation is not ready yet and there were only two nodes available for testing.

Even though more advanced tests were performed, the tests performed should prove that the basic functionality of the implementation works, and may be further developed by others. They may extend Hecuba with e.g., detecting peak hours, closely monitor performance while scaling down, better communication between master and agents etc. Hopefully one day, Hecuba will solve the monitoring and scaling of multiple Cassandra-clusters better and faster than todays database administrators.

11.3 Test cases

Seven test cases will be performed in order to prove that the implementation works as intended. They will also verify that Hecuba have little or no impact on the overall performance of the cluster. In order to do this, the pre-defined test cases are set up to record the resource usage with, and without, Hecuba running. A logger script will record the resource-usage of the nodes. The recorded results from the test cases where Hecuba is running will be compared against the test cases where Hecuba is not running. The results from the comparison will prove if Hecuba works as intended, and if the impact to the overall performance of the cluster is little enough to be accepted. The test cases are assigned a unique id to identify the test when referring to it. An “H” at the end of the name identifies the test cases where Hecuba is running.

TC1_H: No data inserted into the nodes

No data is inserted into the Cassandra cluster in order to record the resource-usage of an empty cluster. This test will record the impact made by Hecuba to the resource usage of the nodes, while the cluster is empty.

The test will also check if there is any bugs that could cause memory-leaks, or any other deviations that could potentially lower the performance, or prevent the Cassandra instance from running as normal.

TC2: No data inserted into the nodes

This test case is almost identical to TC1_H, since it is based on the same preconditions where no data is inserted into the Cassandra cluster. The only difference between TC2 and TC1_H is that Hecuba is not running during TC2. Hecuba will not be running during TC2 in order to record the resource-usage of the nodes when Hecuba is not present. The recorded results from TC2 and TC1_H will be compared in order to determine if Hecuba have an impact on the overall performance or not.

TC3_H: Pre testing, data is inserted into node A, which is automatically distributed to node B

Before the test case will be performed, 1.000.000 writes will be written to node A, which automatically distributes data to node B by Cassandra's built-in functionality. The data will not be equally distributed, as the randomly generated rows will most likely result in a higher density of keys being stored to either of the nodes. Since the insertion and distribution of the data will be performed before the test case is initiated, it will not interfere with the recorded results, and the results will show the node's resource-usage while there is data in the cluster, compared to TC1_H and TC2, where the nodes will be empty. Hecuba will be running during this test, and the results will be compared against the results from TC1_H, since the only difference between TC1_H and TC3_H is that the cluster will contain data during TC3_H.

TC4: Read data from the cluster

Data will be read from the cluster while the logger script records the resource-usage. The goal of this test case is to record the resource usage made by Cassandra while data is read from the cluster. There will be only one client reading data from the cluster, and the recorded results will most likely be different if there are multiple clients reading at the same time.

TC5_H: Read data from the cluster

This test case is almost identical to TC4. The only difference is that Hecuba will be running. Hecuba will be running to be able to record the resource-usage of the nodes with Hecuba active, and later on compare the results against the recorded results from TC4. The comparison will prove if Hecuba has an impact to the performance while data being read from the cluster.

TC6: Insert data into the cluster

During this test, 1.000.000 writes will be written to the cluster. The test will be performed while the resource-usage is logged to see how many resources that are used by Cassandra. The data will be written to the cluster while the test case is performed. The test case is very similar to TC3_H, except that the data will be written to the cluster while the test is performed.

TC7_H: Insert data into the cluster

This test case is almost identical to TC6, except that Hecuba will be active. The resource-usage will be recorded, and compared against the recorded results from TC6. The comparison will prove if Hecuba makes an impact to the overall performance when data is inserted to the node.

11.3.1 The simplicity of the tests

As mentioned above, the tests performed were very simple, and did not last very long. The main reason why the tests were so simple was because the thesis had to be delivered in a few weeks. The current implementation has a lot of small bugs, which prevent heavier and more advanced tests to be performed. A lot of time has been spent understanding the Cassandra code base and try to find the best way to implement the autoscaler. A fair amount of weeks was spent on failed attempts, and when the development of the current implementation started, there was fewer weeks left than expected. A lot of new technology had to be learned as well, which is another reason why the implementation is not completely ready to be deployed to a real cluster. The implementation works even though it is not complete, and may be developed even further for more advanced tests to be performed, and eventually be deployed to a real cluster.

11.4 Results

Linode server 1	
Plan	Linode 2048
Location	New Jersey, USA
Ram	2 GB
Number of CPU's	8*
Storage	48 GB
**The exact amount of CPU-power provided is unknown, as the specification-data was retrieved after the Linode-plans were canceled. The provided data is retrieved from the official Linode website: http://www.linode.com/.*	

Table 11.1: Node A specifications

Linode server 1	
Plan	Linode 1024
Location	London, England
Ram	1 GB
Number of CPU's	8*
Storage	24 GB
**The exact amount of CPU-power provided is unknown, as the specification-data was retrieved after the Linode-plans were canceled. The provided data is retrieved from the official Linode website: http://www.linode.com/.*	

Table 11.2: Node B specifications

The test-results were retrieved from two nodes over a time period of 15 minutes. Node A's specifications are shown in table 11.1 on the preceding page, and node B's specifications are shown in table 11.2. The nodes were located on different continents, node A in USA, and node B in England. There is a time difference of 5 hours between London and New Jersey, which have to be taken into consideration when reading the log-results found in the appendices. After the test cases were completed, the test-result graphs generated, and the node-rental canceled, I noticed that the system clocks were not correctly adjusted. The time difference was 4 hours and 34 minutes, not 5 hours as I first expected. This is not a major issue, even though it is important to remember when reading the appendices.

The logging script that was running did record the node's CPU-, disk-, and memory-usage every fifth second. The logging-script is a separate script found at *bin/logging*, and it is initiated before Hecuba is started. The recorded results showed that the CPU-usage remained very stable during all the test cases. The disk-usage also remained stable, except when a node was scaled up or down. It was not interesting to generate graphs, and go into details about the CPU-, and disk-usage, since it remained very stable, and did not deviate from what is expected when a node is taken down, or brought up. However, if the CPU-, and disk-usage may be of interest, it is found in the appendices at the end of this thesis.

The graphs that visualize the memory-usage of the nodes during the test cases, represents the memory-usage as kilobytes (kB). Since the memory-usage is shown as kilobytes, it will look like the memory usage, drops, and increases, are very high, even though they are not. In most cases it would be better to represent the memory-usage as megabytes, but not when representing the results from our test cases. The impact made to the memory-usage is very little, and therefore it would be hard to show any impact at all if the graphs showed the memory-usage as megabytes.

The y-axis value differs between the different comparisons. If I did not change the kilobyte-scale between the test cases, there would not be possible to describe any changes made to the memory-usage for any

tests except TC6 and TC_7, where the memory-usage were highest. Even though the scale differs between the comparisons, the scale will remain the same for both graphs representing node A and B, in order to preserve the consistency of the comparison. If not, the visual result could have shown something quite different than what was the reality as minor changes could have been represented as huge changes and vice versa.

11.4.1 No data inserted into the nodes

TC1_H & TC2: Two tests, TC1_H and TC2, were performed to record resource statistics from both nodes in order to determine if Hecuba has an impact on the overall performance. The two nodes were empty during both test cases to determine how the cluster behaves initially with, and without, Hecuba running. The tests also gathered more precise recordings of the actual impact from Hecuba, as there were not any interference from third party sources or unnecessary communication between the nodes other than what was necessary. The CPU- and disk-usage was unchanged throughout the tests, while the memory usage was increasing as seen in figure 11.1 on the following page. Figure 11.1a on the next page shows the memory usage of node A during TC1_H and TC2, while figure 11.1b on the following page shows the memory usage of node B during the same tests. Hecuba was running during TC1_H, while it was shut down during TC2.

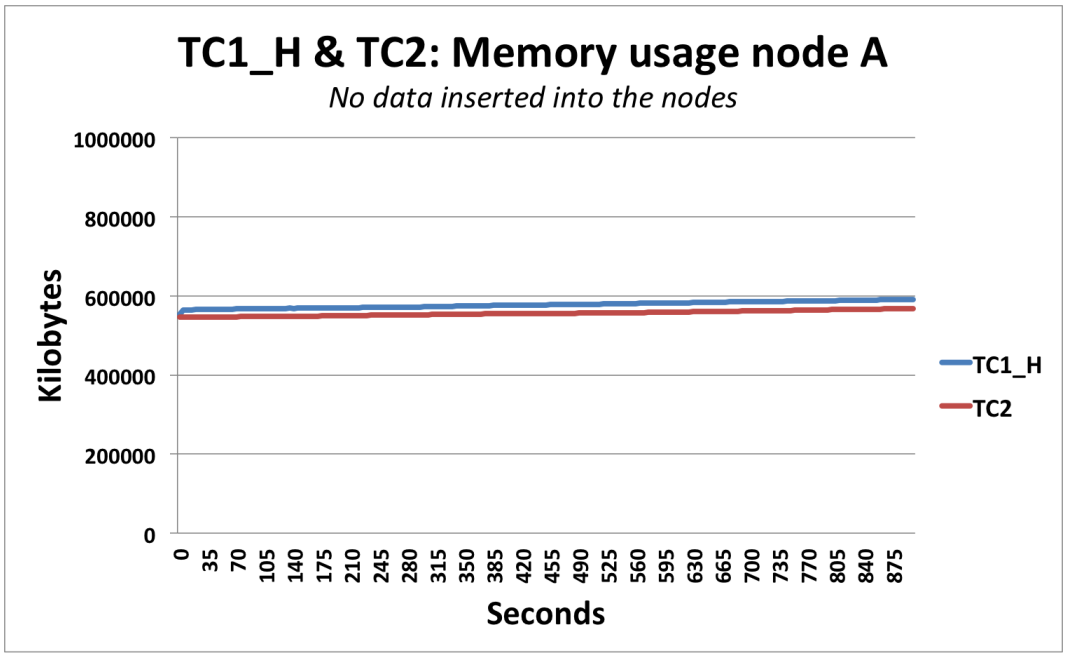
Both graphs show that the memory usage is approximately $20MB$ higher when Hecuba is running, since Hecuba consumes memory in order to run. The sudden spike at the beginning of test 1 was a result of Hecuba being bootstrapped. The logging was initiated before Hecuba, and recorded the bootstrapping as a result of this.

The memory-usage increases steadily throughout both tests, proving that the Hecuba is not responsible for the increasing memory-usage since it increased at the same rate when the implementation was not running. It is hard to determine what generated the memory increase, but it may have been a result of Cassandra executing background processes. Cassandra periodically compact data and keeps as much data as possible in main memory for faster access. Even though the cluster was "empty", there were still a few Kilobytes of meta-data in the cluster that most likely are positioned main memory.

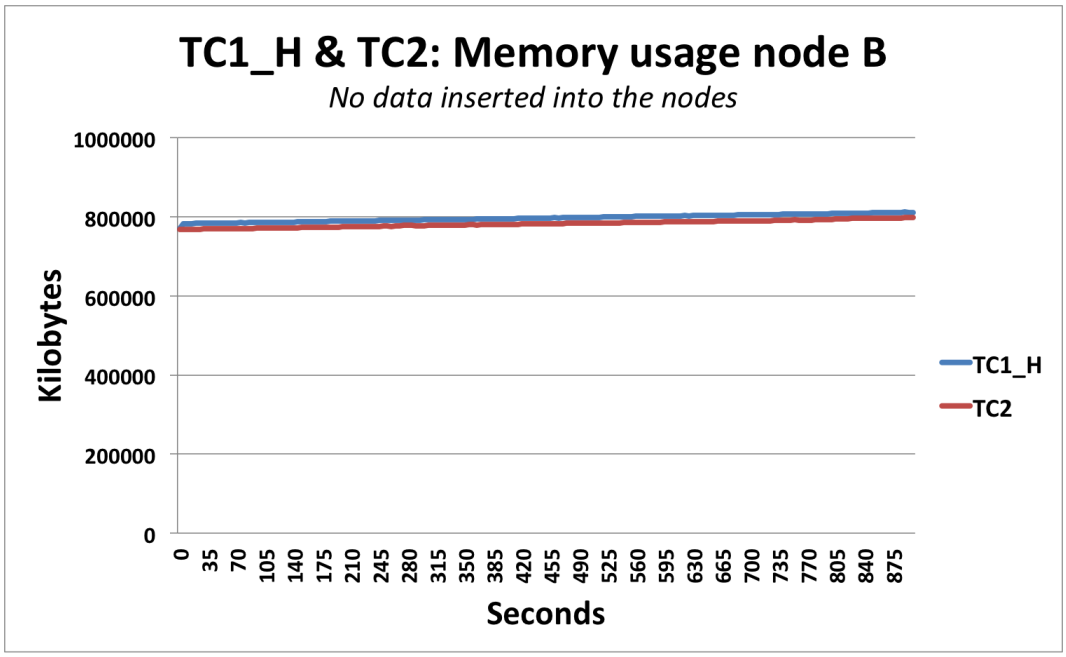
The results prove that Hecuba does not impact the node other than consuming the expected amount of memory in order to run. It is important to remember that the tests are executed on an empty cluster, and the results may be quite different if data is inserted into the cluster or the number of nodes is increased.

11.4.2 Pre testing, data is inserted into node A, which is automatically distributed to node B

TC1_H & TC3_H: During TC3_H, data was inserted into node A, and Cassandra automatically distributed the data to node B. The data was inserted using Cassandra's built-in stress-test tool that randomly generated

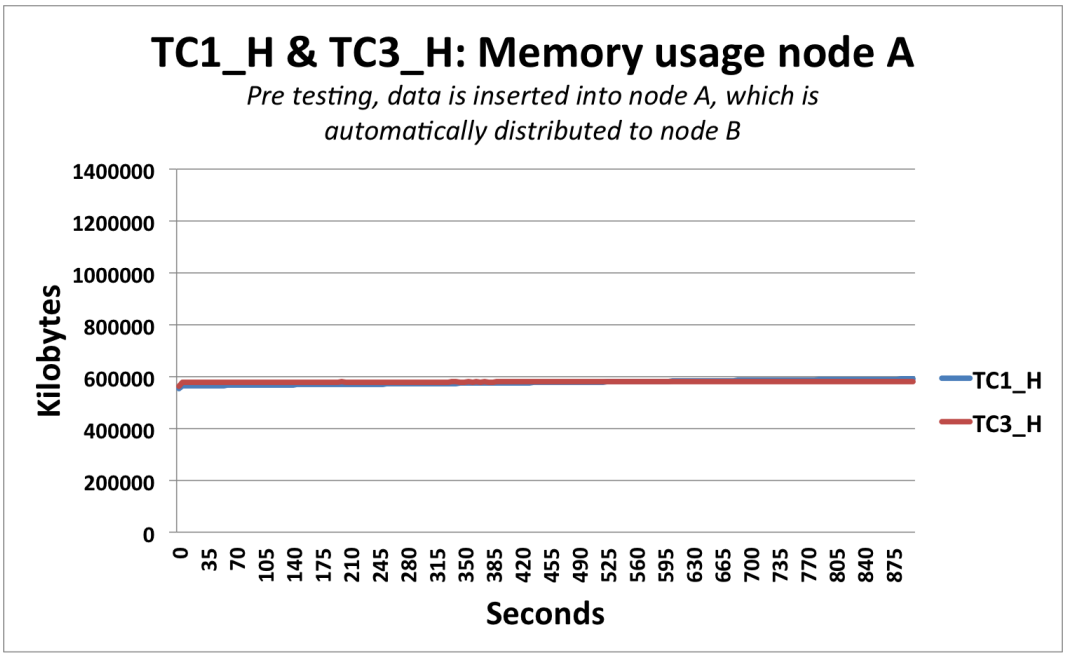


(a) Node A

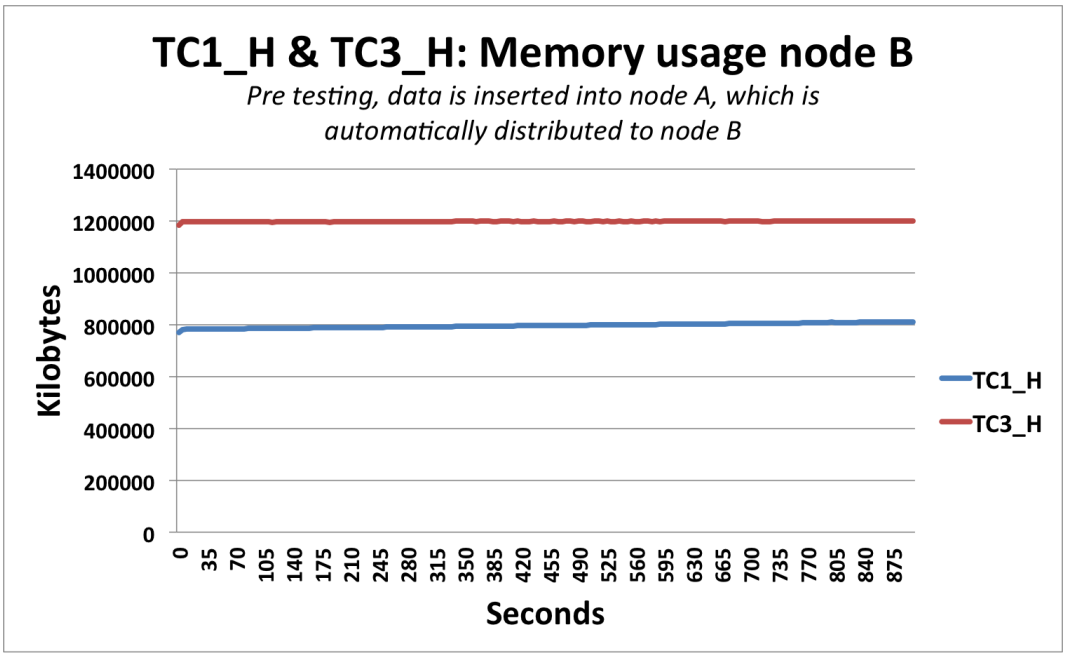


(b) Node B

Figure 11.1: Memory usage during TC1_H andTC2



(a) Node A



(b) Node B

Figure 11.2: Memory usage during during TC1_H compared to TC3_H

rows that was inserted. By default there will be inserted 1.000.000 rows into the node, but it is possible to raise or lower the number of inserts by applying the `-n <number-of-writes>` flag to the stress-test tool.

The data-insertion happened before the tests were started, so it did not interfere with the resource logging. It also had to be inserted into the cluster before the test started to be able to detect how the nodes act when there are data in the cluster, compared to test 1 where there was no data in the cluster.

Figure 11.2 on the previous page shows the memory usage of node A and B during TC3_H compared to TC1_H. In figure 11.2a on the preceding page, node A's memory usage is shown. There was a steady increase in the memory consumption when there was no data available at the node for TC1_H, compared to TC3_H when the node contained $315MB$ of data. A rapid increase in the memory consumption is shown at the beginning of the test, which happened because the logger-script was started before Hecuba was initiated.

Hecuba was running during both tests, which makes them comparable. The only difference between the test cases is that during TC1_H there was no data present in the cluster, while during TC3_3 there was approximately $510MB$ of data present, shared between node A and B. There was approximately $315MB$ of data located at node A, and $196MB$ of data located at node B. The reason why the amount of data is approximate, and why the amount inserted into each node looks very random, is because we used Cassandra's stress-test tool to insert the data. The stress test tool generates random rows, with random lengths, which means that the size of the data inserted may vary, even though the number of writes are the same.

Another interesting observation which is worth mentioning is seen in figure 11.2a on the previous page. The memory usage for TC1_H is initially lower than TC3_H, which is obvious as the cluster was empty during TC1_H. However, when TC3_H's memory-usage stabilizes, TC1_H's memory-usage continues to increase. As seen in figure 11.2a on the preceding page, the memory-usage during both test cases is almost the same. At about 560 seconds, TC1_H uses more memory than TC3_H, even though this is hard to tell from the graphs. This happens even though test 1 does not contain any data other than a few Kilobytes. It is an interesting observation, although there is currently no good explanation for why. It has nothing to do with Hecuba, as TC1_H and TC2's memory-usage increases at the same rate as seen in figure 11.1a on page 90. It was either some external processes or an internal process initiated by Cassandra. Since a third party process may have interfered with the test results, the tests should have been initiated more than once, and an average of the results should have been calculated. Due to the lack of time, the tests were only initiated once, and therefore the results may contain some external "noise".

Figure 11.2b on the preceding page shows the memory usage of node B during TC3_H compared to TC1_H. Both tests show that the memory usage was very stable with, and without, Hecuba running. The only difference is that during TC3_H, the memory usage was higher. The reason why the memory usage was higher during this test is because there was data present

in the cluster that was present in memory as well. Cassandra tries to hold as much data as possible present in main memory in order to keep the performance as high as possible. Since TC3_H only inserted about $510MB$ of data, Cassandra did not have problems keeping all of it in main memory. Since all data was kept in main memory, the memory usage increased. If the data was written to disk, and the memory flushed, the memory-usage would have decreased.

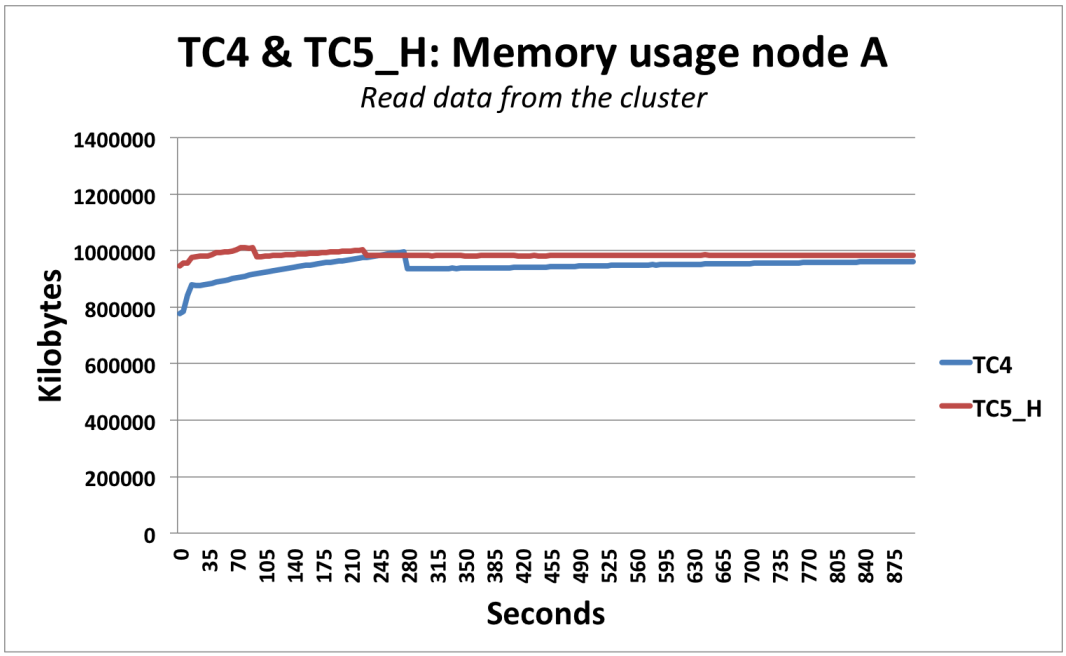
Apart from the memory consumption, the disk- and CPU-usage was very stable during both tests. Since there were not any huge deviations, the disk- and CPU-usage for these tests will not be further explained. The CPU-, and disk-usage may be found in the appendices for the appropriate tests.

11.4.3 Read data from the cluster

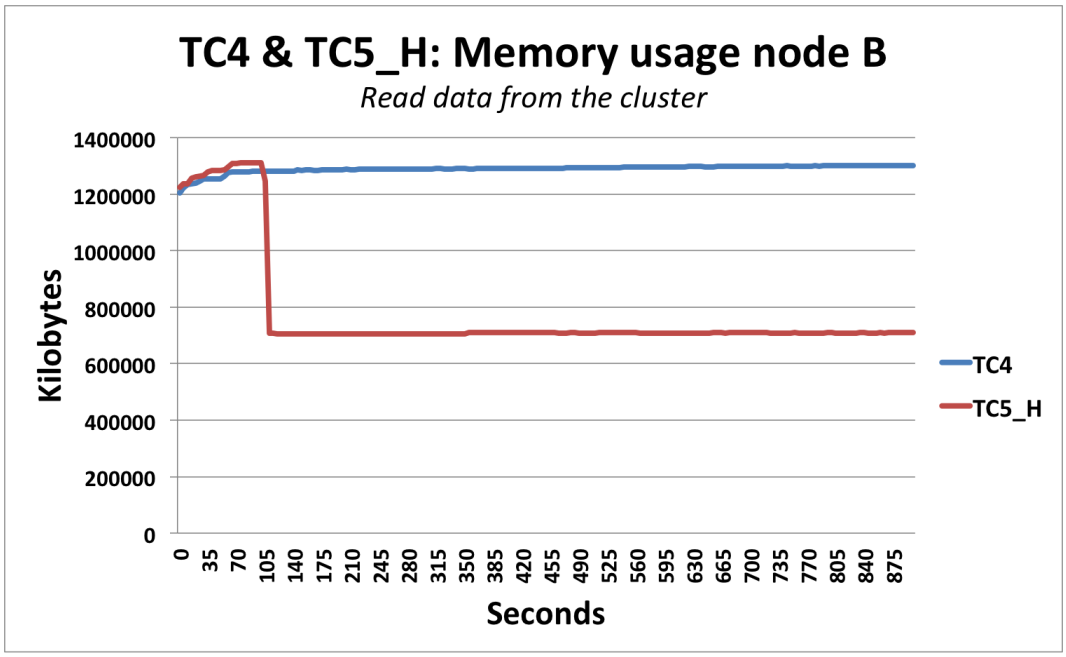
TC4 & TC5_H: During these tests, node A contained approximately $315MB$ of data, while node B contained approximately $196MB$ of data. The data present is the same data that was inserted into the cluster before TC3_H was performed. TC4 was executed without Hecuba running. Hecuba was shut down to make sure both nodes stayed active within the cluster, and to see the resource-usage made by Cassandra, without Hecuba interfering. Hecuba was running during TC5_H, to see if the implementation caused any performance loss or not compared to TC4. TC5_H had the minimum disk usage set to $200MB$, which lead to node B being scaled down while the test was running.

Figure 11.3 on the next page shows the memory usage of node A and B during TC4 and TC5_H. The only difference between the tests was that Hecuba was not running during TC4, while it was running during TC5_H. Figure 11.3a on the following page shows the memory usage for node A, while the tests were running. At the beginning of the tests, the logger recorded the initial startup of Hecuba, shown as a minor increase in the memory-usage for TC4. The memory-usage continued to increase, which was a result of the read-requests that was directed to the node. After about 280 seconds, the memory-usage drops, and stabilizes. This happened because the read-requests finished.

The memory usage during TC5_H was a bit unstable at the beginning of the test, compared to TC4. It was also a bit higher for TC5_H than TC4, which was because Hecuba was running. A downscale of node B was also initiated at the beginning of TC5_H. The scale down may be seen in figure 11.3b on the next page. As shown in the graph, the memory-usage stabilizes when the data received from node B has been transferred and committed locally. When the memory usage was stabilized during TC5_H, it did not continue to slowly increase as shown for TC4. This probably did not happen because there was only one active node in the cluster, which leads to no communication with other nodes in the cluster. There were two active nodes that communicated and shared meta-data during TC4, which may have generated the increase in memory-usage. It may also have been external processes that were running on the nodes, even though this minor



(a) Node A



(b) Node B

Figure 11.3: Memory usage during TC4 and TC5_H

increase is not important for the final result.

Node B's memory usage during TC4 and TC5_H is shown in figure 11.3b on the facing page. The memory-usage increases at the same rate for both tests at the beginning. The reason why TC5_H's memory-usage is a bit higher is because Hecuba is running. After 70 seconds, the memory-usage for TC4 stabilizes, while it stabilizes after approximately 105 seconds for TC5_H. It took longer time for TC5_H to stabilize because the node was scaling down, which require some extra memory-capacity.

Since the majority of the data were located at node A, the majority of the read-requests were also directed to node A. This lead to fewer read-requests, and less activity for node B, which is the reason why there is no visible activity for this node. By looking at TC5_H, the memory usage clearly shows when the node was scaled down. The memory usage increased at the same rate for both tests at the beginning, which probably was a result of the data being read from the cluster. The read-requests ended early in TC5_H, which is shown by the memory usage stabilizing. After 105 seconds, the memory usage for this test experienced a memory-usage drop at node B. This is a result of the node been decommissioned, and removed from the cluster. The memory-usage stabilizes directly after the drop. The Cassandra instance is not running anymore, which is the reason why the memory-usage during TC5_H at node B is a bit lower than during TC4.

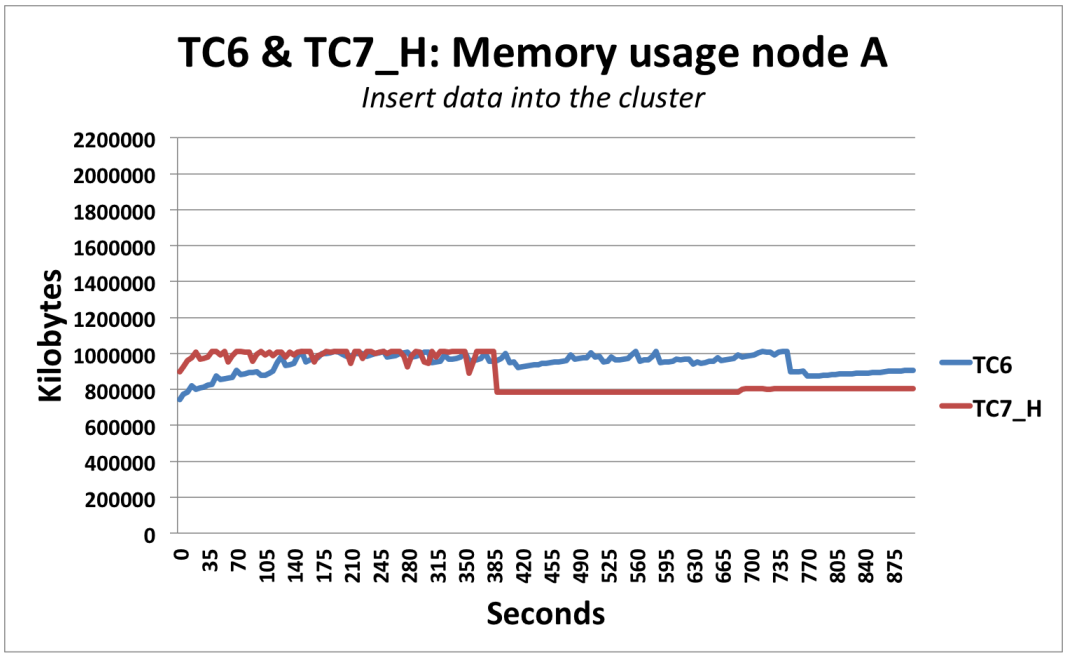
The test results are very similar, except that node B was scaled down. The beginning of the test is almost identical, and when the memory usage stabilize, the usage follows the same pattern with, and without Hecuba running, which proves that the implementation does not have an impact on the cluster while data being read.

The CPU- and disk-usage did not deviate from the expected results, and therefore they will not be described in detail. The disk usage of node A had a huge increase during TC5_H, which is a result of node B being shut down, and data moved from node B to node A. For detailed information about the CPU-, and disk-usage, see the appendices for the appropriate tests.

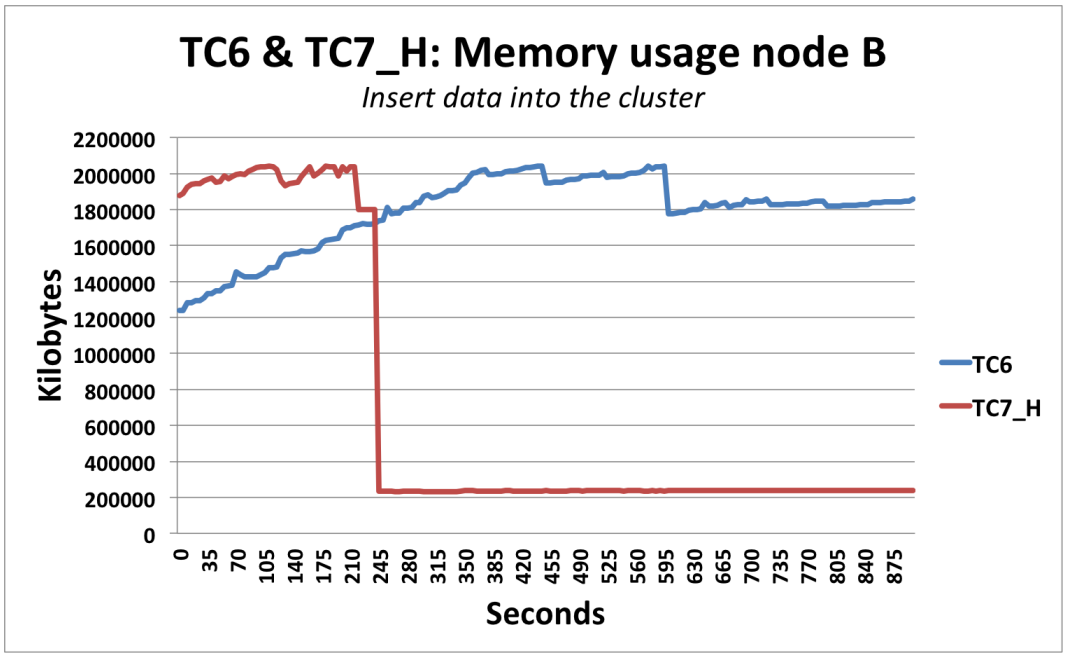
11.4.4 Insert data into the cluster

TC6 & TC7_H: Figure 11.4 on the next page shows the memory usage of node A and B during TC6 and TC7_H. Hecuba's impact on the cluster while data being inserted into the cluster was tested in order to see if there was any deviations from the normal behavior that could lower the overall performance. Hecuba was shut down during TC6, and running during TC7_H to see the difference of the resource usage with, and without Hecuba running.

Node A's memory usage during both tests is shown in figure 11.4a on the following page. The data was inserted through the Cassandra stress test script at node A. The script inserted data until for 700 seconds. It is hard to determine exactly when the insertion ended, as the graph drops after 595 seconds for node B, and 735 seconds for node A during TC6. The memory-usage drops around 385 seconds for TC7_H, which happened because the



(a) Node A



(b) Node B

Figure 11.4: Memory usage during TC6 and TC7_H

insert-script stopped, node B were scaled down, and the data received from node B was stored locally. TC6 shows that there was an increase in the memory-usage at the beginning of the test. The memory-increase experienced memory-spikes, which is a result of data being inserted into the node. Data-insertion requires the node to calculate the position in the ring for the inserted data set, and place it at the correct location. The location in the ring may be at node B, which requires node A to transfer the data set to node B. There were memory-spikes until approximately 700 seconds, when the memory usage dropped and stabilized compared too earlier in the test. The memory usage dropped because the insert script ended. Hecuba was not running during TC6, which is the reason why no signs of a scale up or down at either of the nodes is shown.

Hecuba was running during TC7_H. Figure 11.4a on the preceding page shows the memory usage for node A during TC6 and TC7_H. It had a bit higher memory-usage than TC6, which is a result of Hecuba running. Around 175 seconds, both test cases uses the same amount of memory, and follows the same memory-spike pattern. Since both test cases follow the same spike-pattern, we can be very certain about the spikes coming from the insertion-script, and is not a result of Hecuba running. TC7_H's memory consumption dropped when the test had been running for 385 seconds. This probably happened because the insertion script ended, and node B scaled down. The results from node B seen in figure 11.4b on the facing page shows that the memory-usage for TC7_H dropped $385 - 245 = 140$ seconds later for node A, compared to node B, where it dropped after 245 seconds.

Compared to the results from node B, seen figure 11.4b on the preceding page, it scaled down a bit earlier than the memory drop at node A. When the memory dropped at node B, the data was already transferred to node A. Node A continued to experience memory-spikes since the insertion script was still running, and data was received from node B. The data that were received was reorganized at node A before it could be stored. Since node A reorganized about $480MB$ of data from node B, the spikes continued after the insertion script ended. When the data was successfully included into the existing data at node A, the memory usage dropped and stabilized. By looking at the graphs for node A and B, the memory-usage seems to keep increasing when TC6 ended. The insertion-script had ended, so the memory-increase probably occurred because Cassandra was sharing data and replicas between the nodes.

Node B's memory usage is shown in figure 11.4b on the facing page. The graph shows less memory-spikes than for node A, which may be a result of node B receiving less data than node A. Node A was responsible for a larger amount of the overall data during the tests than node B, and therefore most of the activity happened at node A. TC6's memory usage increased steadily until about 735 seconds. Then the usage dropped, because the insert-script ended before the test was completed. There are not any visible spikes in the overall memory usage at node B, which may be a result of the majority of the data being sent directly to node A.

It is easy to see when node B was scaled down during TC7_H, as there

were a huge drop in the memory usage after 245 seconds. The memory usage for TC7_H is higher at the beginning than for TC6, which probably were because Hecuba was running and sending messages to the master. When TC6 had been running for 245 seconds, the memory-usage dropped as the node was scaled down. The memory-usage stabilizes after the memory-drop, since Cassandra was shut down and the memory used by Cassandra were freed. Hecuba continued to send breach messages, since the current implementation does not stop sending messages when the node is scaled down. A future release of the implementation should prevent the agent from sending breach messages when the node is scaled down.

Hecuba does not have a noticeable effect on the overall performance since it is running when the node is scaled down, and the memory usage is stabilized as seen for test in figure 11.4b on page 96. It is worth mentioning that the implementation have been tested with simple test cases, and that the result may be quite different if there is more data in the cluster, more nodes, more complicated tests performed etc.

There are no deviations in the results for either the disk- or the CPU-usage. The CPU-usage stays stable during both tests, and only deviates 1% from the start to the beginning of the test, which is seen in the appendices for TC6 and TC7_H. The disk usage for both nodes steadily increases until node B is scaled down. The disk space for node A continues to increase after the disk space dropped for node B, as data is being received. The newly received data also has to be compacted and inserted into the database, which generates extra Meta-data and possible duplicates. The amount of data at node A also decreases after a while, as unnecessary data-duplicates and meta-data is removed. It took 145 seconds from when node B removed its $535MB$ of data, until the data amount stabilized at $911MB$ at node A. Node A was containing more than $911MB$ of data at some point, as there were probably duplicates of data received from node B that had to be taken care of.

There have not been generated any graphs from the disk-, or CPU-usage, since the recorded results does not deviate from what was expected. However, the data is found in the appendices for TC6 and TC7_H at the end of this thesis.

11.5 Summary

Seven tests have been executed to prove that Hecuba works as intended. They have tested that Hecuba does not impact the overall performance of the nodes when there is data, and where there is no data, available in the cluster. The tests have also verified that Hecuba does not impact the performance while data is being inserted into, and read from, the cluster. There have been performed two tests to verify that Hecuba does not impact the nodes: one test where Hecuba is running, and another test where Hecuba is not running.

The tests were performed without anything-unexpected happening. The results satisfies the initial goals for the test cases; that Hecuba should

not impact the overall performance of the cluster, and the cluster should be running as if Hecuba did not exist. Even though there was performed a set of simple test cases, they proved that our implementation of an automatic scaler for Cassandra works, and it may potentially be a success if it is developed even further.

Part IV

Conclusions

Chapter 12

Assessment of Hecuba

12.1 The Design

I think that Hecuba is a success so far, considering the limited amount of time available for such a large project. There are some faults that maybe should have been taken care of, but the overall result satisfies my goal for this thesis. Hecuba's current design was not my only idea of how to implement the automatic scaler.

Firstly I developed a design that was supposed to be implemented directly into Cassandra's source code. In the beginning, when I did not have enough knowledge of how Cassandra was built, I thought this was a very good idea. The idea was to implement Hecuba as a standalone daemon that launched together with the Cassandra daemon. It would also take advantage of the gossip protocol in order to communicate with the other nodes. The design could have become a very good solution, where it would not be necessary to install any third party extension to Cassandra. It would have taken advantage of the already existing *gossip protocol*, which is already implemented into Cassandra's. However, I did not think of the maintenance-work that would follow if I modified Cassandra's code base. I would have to re-implement the autoscaler every time a new version of Cassandra was released, if not the developer-team for Cassandra merged Hecuba into the official releases. However, this would most likely never happen. I should have foreseen that modifying Cassandra's code base would require a lot of maintenance work, and probably was not the best solution, as already existing clusters would have to be re-deployed with the modified Cassandra version.

The second attempt was similar to the final result, except the majority of the functionality were located at the master, and it was very tightened to Cassandra. The daemon that monitored the resource-usage were located at the master, which means that the master were sending messages to its agents every nth second, asking for their disk-, CPU-, and memory-usage. This happened even if there were no breaches occurring. This required a high amount of data to be sent forth and back between the master and its agents. More data sent across the Internet means that the master would have to wait for messages to be received, since it takes time

to send messages across the Internet. There will also be a higher change of failures to occur. The majority of the resource-values are not of interest to the master, as the master is only interested in the resource-values when a breach occurs. The constant flow of messages being sent across the Internet is the main reason why the second attempt experienced major changes. I moved the monitoring from the master to the agents, and the agents would only send messages to the master if a breach occurs. By doing this, I lowered the amount of messages being sent from each agent. If I set the monitor to check the resource-values every second, and send a message to the master if a breach occur more than ten times in a row, I would have lowered the network traffic between the master and its agents by 90%.

Towards the end of my work with the final solution, I came up with many new ideas that would have improved Hecuba. Among other things, I wished to implement smarter node-insertion, automatically re- balancing of the cluster, and automatically receive the list of current active nodes upon agent-initialization. Since the deadline of this thesis was closing in, I had to finish my current solution in order to perform tests proving that my implementation worked. I started to develop Hecuba from scratch, without any knowledge of distributed systems, Cassandra, or automatic scaling. I have described my ideas of what may be done to further develop Hecuba, since I did not manage to do it all myself. I hope these ideas may be of help to anyone wanting to continue my work for an automatic scaler implementation.

In my current solution, there are a few design-issues that I wish I had time to correct. The Autoscale-agent continues to send breach-messages to the master after it has been scaled down. The master will collect the messages, and include the nodes into the calculation of which node that should be scaled up or down. The outcome of the scaling-algorithm may become a node that is already scaled down, that is told to scale down again. This will not result in an immediate problem, but it may become a problem if the node that should have been removed is not. However, since it is scaling down, it is not as urgent as if it were scaling up. On the other hand, if the scaling-algorithm resulted in a node being scaled up, that already were scaled up, the outcome would have been quite different. It could have resulted in the cluster loosing data if it runs out of disk-space. Users could also experience poor performance, since the cluster does not have the necessary resources available.

12.2 The Implementation

I am satisfied with the final result. I am glad I did not implement it directly into Cassandra's code-base, since it would have been a lot of maintenance work. I have not performed any tests proving that the implementation works over a longer period of time, meaning it cannot be deployed to a live cluster yet. Even though I did not manage to perform such tests, I managed to develop an implementation that runs, and where the scaling occurs. There are a lot of improvements that I would have liked to do

before Hecuba is released. Most importantly, I would like to prevent the agents from sending breach messages to the master after the agent has been scaled down. This should not be a lot of work, although I noticed this after the tests were completed. If I were to make changes to the source code, I would have to perform the tests all over again, in order for the results to match the current implementation. I could also have encountered different obstacles, which could have consumed a lot of the remaining time for this thesis. Apart from this, I think that the implementation may potentially work if it were deployed to a real cluster, however I would not recommend anyone doing it before Hecuba is fully tested.

The resource-usage of the current implementation is very low. It requires approximately 20MB of RAM, which may be seen in the test-results by comparing e.g., TC1_H and TC2 (see appendices). This is not much at all, and is only the amount of ram necessary for Hecuba to run. Hecuba does not affect the overall performance of the nodes, which is what I expected. However, it may generate a lot of memory-, and CPU-usage if there is a lot of data within the cluster, and if there are nodes that are being scaled up and down frequently. I have not yet tested this, but the Cassandra instance will have to move data between the nodes in the cluster when nodes new nodes are inserted, and existing nodes are removed. This is expensive operation, as data first have to be transferred across the Internet, before it have to be compacted, reorganized and indexed at the new node. The reason why I did not test this was because I did not have the necessary amount of nodes available, and I also suspect the implementation to fail if it is put to such test. Hecuba does not wait for nodes to successfully scale up or down, meaning that it may potentially tell a node to scale down before it has been completely scaled up. I do not know what will happen then, but I am afraid that data may be lost, and the overall performance of the cluster will probably suffer.

Hecuba was not straightforward to implement. I had a lot of *"try-and-fail"* attempts, as I had never developed a Java-application that was split into several different projects, where the applications communicated together. I had also never accessed the local resources of the machine where the applications are running. I had to use most of the time to understand how this was done. I had also never developed a Java-application that was launched from a script, as Cassandra is. Even though I had never created that kind of script before, I looked at Cassandra's startup-script, which I used as inspiration to create Hecuba's startup-script.

I mean that the current implementation works, as the scaling is triggered and successfully performed. However, it does not work properly, as there are known bugs that have to be fixed before it is officially released. If the bugs are fixed, and the implementation is further developed, the test cases that are performed should also be further developed. There should also be developed a test-suite, containing the tests that should be performed, any data that is required for the tests to run, and the execution-order of the tests. There should be developed tests that runs for a longer time than the tests I performed. There should also be developed tests that are triggered from multiple nodes, triggering up and downscaling

randomly. Least but not last, tests should be developed that generate incoming traffic to the nodes in order to see how the implementation performs when there are activity within the cluster.

The current implementation stops working if the cluster demands new nodes, and the list of inactive nodes are empty. This is a known issue, which may be solved by e.g., the database administrator initializing new nodes into the cluster manually, or extending the implementation with a script that initialize nodes at e.g., Amazon Web Services. I did not think of this while working with the implementation, as I never encountered the problem. If the increase of nodes comes from a temporarily increase, which will stabilize and go back to normal again, it should be enough if the database administrator fires up new nodes whenever needed. If the increase comes from a sudden increase of interest for your product, it may be desirable that a script is executed, which triggers initialization of new nodes at e.g., Amazon Web Services. Amazon Web Services have an API that allows such scripts to be performed. However, I think that this may not be necessary, as most of the clusters that are available today should have the necessary amount of nodes available. Hecuba will stabilize the number of active nodes in the cluster, in order for the cluster to run as few nodes as possible, without lowering the performance. This means that Hecuba may scale down a few nodes directly after it has been deployed, but it will stabilize when it reaches a certain point. When it stabilized, it will wait for an increase or drop in the memory-, CPU-, or disk-usage. If the database administrator are afraid that there is too few nodes available for Hecuba, he or she should initialize a couple of extra nodes to make sure that the implementation will not run out of nodes. Hecuba should implement functionality to alert the database administrator whenever there are fewer than n number of nodes left in the list of available nodes, e.g., 1 node available for insertion.

Chapter 13

Test analysis

The test cases I performed were simple and short, and are therefore not enough to determine if Hecuba is robust and stable. There should be performed more advanced test cases that consist of a larger amount of data, and randomly generated network-traffic towards the cluster. The test cases should also run for a longer time, and consists of a higher number of nodes.

The main reason why the test cases were so simple was because I did not have enough time to further develop Hecuba, and create the test cases. As soon as Hecuba were ready for testing, there was not enough time left for developing more advanced test cases that I knew would generate useful statistics. However, I think that the test cases performed prove that the implementation are capable of scaling, and does not impact the performance. The test cases were also performed on two nodes. It does not reflect a real cluster, but I think it is enough to prove that the scaling occurs. I do not think the outcome would have been any different if there were more nodes in the cluster.

13.1 Comparison

We have performed seven test cases in order to verify that Hecuba works, and that it does not impact the overall performance of a Cassandra cluster by consuming too many resources. All tests may not be compared directly against each other, since they have different pre- conditions. However, some of the tests have been performed with the same pre-conditions, where the only difference was that Hecuba was running for one of the tests, while it was shut down for the other. This was done in order to determine if Hecuba had an impact on the resource-usage, and the overall performance.

The first two tests, TC1_H and TC2, proved that Hecuba did not have an impact on the performance if the cluster was empty. As shown in figure 11.1 on page 90, the memory usage was not very different for the two tests, except that the amount of memory used was a bit higher for TC1_H than TC2. TC1_H used a bit more memory than TC2 since Hecuba was running. Other than that, the steady increase of memory usage was very similar for both tests. The CPU- and disk-usage was monitored as well, but the recorded values are not of interest, as they did not change throughout

the tests.

The third test, TC3_H, proved that Hecuba did not affect the overall performance while there was data available in the cluster. Data were inserted into node A, which automatically distributed the data to node B, before the test case was initiated. The memory usage is very stable for node B, but it is a bit higher than for node A. It is higher because Cassandra holds the local data in main memory for faster access. Node A shows that the memory usage is more stabilized for TC3_H than TC1_H, even though the cluster was empty during TC1_H and contained data during TC3_H. The memory usage at the end of TC1_H is even higher than at the end of TC3_H. It is not easy to say what generated the steady increase of memory usage for TC1_H. Even though we do not have an answer to the memory increase of this test, it cannot be Hecuba. The memory-usage increases at the same rate with, and without, Hecuba running as seen in the figures 11.2 on page 91.

TC4 and TC5_H proved that Hecuba did not have an affect on the overall performance while data was read from the cluster. The data inserted for TC3_H was still in the cluster during these tests, because the tests should be performed on a cluster containing data. The Cassandra stress test tool was used to generate random reads from the cluster while the tests was running. Figure 11.3 on page 94 shows the memory usage of TC4 and TC5_H. Hecuba was running during TC5_H, and shut down during TC4. Both tests have an increase in the initial memory usage at node B. A few seconds after the test started, the memory usage stabilize for TC4, and stays stable throughout the test. This probably happened because Hecuba was not running, and the data-insertion had stopped. TC5_H shows that directly after the memory usage stabilized, there was a drop in the overall memory usage. Since Cassandra was shut down, the data held in main memory was removed. As the Cassandra instance and its data were removed from main memory, the overall memory usage decreased as shown in figure 11.3b on page 94.

The memory-usage for TC4 increased rapidly at the beginning, but stabilized when the read-script ended after 280 seconds. TC5_H had a bit higher memory consumption, even though it was more stable than for TC4. It is not clear when the read script ended, as the first part of the test contain both read-requests and data received from node B. Data was received from node B since it was scaled down. After the read script ended and the data from node B was organized and stored, the memory usage stabilized at higher level than for TC4, which was because our implementation were running.

The sixth and seventh tests, TC6 and TC7_H, were the last tests that I performed, and the memory usage is shown in figure 11.4 on page 96. Data was inserted into the cluster while these tests was running in order to see if Hecuba have an impact on the performance while data is being inserted into the cluster. Hecuba was not running during TC6, and the insert-script was running until the test was almost complete. The majority of the inserted data was inserted into node A. Figure 11.4a on page 96 shows the memory usage of node A, and there is a lot more activity at

this node than node B which is shown in figure 11.4b on page 96.

Hecuba was running during TC7_H, where node B was scaled down after the test had been running for approximately 245 seconds. The memory usage of node B drops a few seconds before it drops at node A. This may be because node A needs some time to reorganize the received data, but it may also be that the insert-script was not finished. After node B was completely scaled down, and the data reorganized and stored at node A, the memory usage of both nodes were sustained throughout the rest of the test.

As for the previous test results, the CPU-, and disk-usage did not deviate enough to be of interest. The disk-space was the only attribute that did change, and is worth mentioning, even though it changed as expected according to the downscaling. Since there was not much to say about these changes, there was not generated any graphs to visualize the results. The results that were logged may be found in the appendices for TC6 and TC7_H.

13.2 Discussion

The focus of the tests has been to make a *"proof of concept"* that Hecuba work, without interfering with the already existing Cassandra implementation. Even though the tests were simple, they did prove that Hecuba is initialized, and successfully completes a downscale without affecting the cluster's performance. The results may have been different if there was more data or more nodes involved, although this was not the goal of these tests as they was supposed to prove that Hecuba initiate the scaling process.

We have been focusing on the memory usage of the nodes as it best showed the different happenings throughout the tests. Cassandra holds as much data as possible in memory to respond to requests fast. It also performs its calculations and reorganizing in memory, which is shown in some of the recorded results. It also made sense to focus on the memory usage since Cassandra works mainly in memory, and any operations or deviations by Cassandra would be reflected to the recorded results and shown in the graphs.

It is not possible to conclude if the implementation is a success or not, since the tests did not test any real scenarios. The tests were not performed on a realistic cluster, and they were only running for a short period of time. In order to come up with a conclusion if the implementation is a success or not, it should be further developed with functionality to detect low-peak hours for scaling, more controlled scale-down of nodes, stop sending messages if the node is scaled down etc. The bugs that exist should also be fixed, and Hecuba should be put to test by more complex tests. The tests should last longer, in order to see how Hecuba acts when multiple scales occurs.

Even though it is hard to say if the implementation is a success or not, it is clearly a good start. The tests prove that the implementation is able to

scale and continue running. The tests also show that the implementation does not affect the resource usage more than necessary in order to run.

Chapter 14

Summary

The idea of this thesis was to develop an implementation that was able to automatically scale Cassandra clusters. There have been several changes to the original design, which resulted in the implementation as it is today. It was originally supposed to be implemented directly into Cassandra's source code, and take advantage of the already existing gossip protocol. This design-idea changed during the thesis work, since we learned more about the technology used, and found better solutions to the problems that occurred. The final result ended up as a standalone implementation where there are only a few references to Cassandra itself. Another factor that led us towards a standalone implementation is that we did not have to develop, and update our own fork of Cassandra's source code. This would result in a lot of work in order to keep up with new releases, which would most likely result in our Cassandra-version being old and outdated.

We performed *proof-of-concept* tests in order to prove that the scaling occurs, and that it is triggered based on a set of pre-defined values. Most of the test cases were performed twice where the implementation was shut down during one of them in order to determine the impact of the implementation to the local node's resources. Different actions were performed while the test cases were monitoring the resource-usage in order to see if the implementation could potentially lower the performance. If the implementation lowered the overall performance of the Cassandra-cluster, it would be useless and could not be deployed to a real cluster. The tests monitored the resources while the cluster was empty, consisted of data, data was read, and written.

The test cases performed are not enough to be able to come up with a final conclusion if the implementation works or not. However, the tests prove that Hecuba's currently developed functionally work as intended. There are some bugs that still exist, but the implementation still works. They also indicates that Hecuba does not acquire more resources than necessary in order to run, which means that Hecuba may be deployed to an already existing cluster without affecting the overall performance too much. The only problem is that the scaling-algorithm may trigger the scaling-process too often, resulting in a lot of extra activity in the cluster. Apart from the known bugs, the implementation may become a success if

it is further developed to handle e.g., peak-hours, controlled down scaling, and more intelligent up scaling, where the inserted nodes tries to balance the cluster as much as possible to prevent skewness by calculating a more suitable location in the ring.

Today the implementation is immature and a working in progress. There are still bugs that have to be fixed and important functionality to implement before the implementation is ready to be deployed to a real cluster. There is uncertainty about the test results since there was not performed more complex tests. More complex tests should have tested how the implementation react to multiple active connections made to the cluster, peak-hours, a larger amount of nodes, and more data in the cluster. Such tests were not performed because there was not enough time left of the thesis work. The implementation is not mature enough yet, which would result in a lot of bug fixing and developing in order to perform the tests.

Hecuba has some known problems that may prevent more complex tests to be performed. It does not stop sending breach messages after a node has been scaled down. This is not an important bug, but it may cause problems when e.g., the master marks the node to be scaled down the 2nd, 3rd, or 4th time. A node with high resource-activity that is already scaled down will also prevent active nodes from being scaled up or down, which may potentially prevent important scale- ups to occur. A more important bug is that the shutdown command does not completely kill the Cassandra process at this point. The same command that is written to the command line in order to kill the running Cassandra process is executed, but the instance is still running. Even though the Cassandra instance is running, the node is decommissioned and removed from the cluster, and the content of the data-folder is removed. Since the tests performed was so simple, and did not trigger both scale up and down, it is hard to say if this could cause problems if a node is shut down, and brought up again later on.

Even though we cannot give a final conclusion whether the implementation is a success or not if it were deployed to a real cluster, we think it will be if it is further developed. We feel that Hecuba may compete with Priam one day since it solves the scaling-problem differently, as well as proving functionality that is not available in Priam. On the other hand, Priam provides functionality that is not available in Hecuba. Since both solutions provide functionality the other does not, we think that Hecuba may be a success in its own area: if someone is in need of both up-, and downscaling. A lot of work have been put into completing the implementation on time, and provide the desired functionality. It may currently be used for testing purposes, and we encourage others to create different test cases in order to get a broader set of results that hopefully will result in improvements to the implementation.

14.1 Further work

The outcome of this thesis is a simple automatic scaler for Cassandra-clusters, named Hecuba. It does not consider low-peak times, controlled

up- or downscaling, smart insertion of nodes etc. A more advanced, and complete implementation is required in order to fully satisfy the requirements of an automatic scaler. The test-results have shown that it may potentially be a success, however it is hard to predict how the implementation behaves when a cluster consists of more data, a higher number of nodes, multiple up- and downscales occurring etc. It would be interesting to see how the implementation would perform on a more realistic scenario e.g., a cloned version of one of Netflix's Cassandra clusters[29].

In the following subsections we summarize different areas applicable for further work.

A more generic implementation

The current implementation is partly generalized. There are still a few places within the source code where Cassandra-specific code can be found. The AutoscaleDaemon class' init-method initiates the CassandraHostManager class. The CassandraHostManager implements the HostManager interface, which interfaces all the required methods. It should be an easy task to move the HostManager implementation-path that are being used to e.g., a separate configuration file. Currently the master-implementation does not have a configuration-file. Therefore, it has to be extended with both a configuration file, and a reader-implementation to successfully load the attributes from the file into memory. When we have successfully created the properties-file, and the reader-implementation, an attribute for the HostManager may be added, e.g., `host_manager = no.uio.master.autoscale.host.CassandraHostManager`. The reader will create a java-object from the path provided. This is achieved by executing the Class.forName(String className) method, which would return a Cassandra- HostManager object (It must be casted to HostManager). This way, another HostManager may be applied to the implementation by appending it to the class path, change the value of the host_manager attribute, and restart the application.

The implementation is currently designed to run on Linux-machines only. Linux-specific commands are executed while starting and shutting down the Cassandra instance. These commands should be replaced by Java-code if possible, or a set of commands for all major operating systems should be provided. Which command-set the implementation are using should be automatically detected in order for the users to deploy Hecuba to machines without having to know which Operating System that is currently running.

Infinite number of available nodes

The number of nodes available are currently limited by the actual number of nodes that have been initiated to the cluster. E.g., if there is only initiated five nodes to a cluster, the cluster will not be able to scale up to more than five nodes. This means that the database administrator have to initialize

all nodes that should, or may be, a part of the cluster. The autoscaler will downscale the cluster to a minimum number of nodes required in order to provide good performance, but not less than the minimum number of nodes set by the configuration property `min_number_of_nodes`.

All nodes that are supposed to be a part of the cluster do not have to be initialized during startup. The master should occasionally ask a random agent for a list of currently active nodes, which will include any new nodes that have been added to the cluster after the initial startup. The current implementation does not have this functionality, even though it is highly recommended for an automatic scaler in order to be fully automatic. The master implementation will initiate an agent-implementation on each new node, which will lead to the node's agent starting to monitor, and sending feedback to the master. The current implementation will not be able to initiate new nodes if the list of currently inactive nodes is empty. If the traffic and the amount of data continue to increase, the agents will continue to send breach-messages to the master, without the master being able to do anything.

If this happens, and the database administrator is not available, the implementation will run into problems. The current implementation does not handle this. A solution to this may either be to alert the database administrator through Email, SMS, etc., or to solve the problem without human interaction, even though this is not preferred. To exclude all necessities of human interaction, Hecuba may be extended to support *"infinite"* number of nodes. This will only be possible if the cluster is running on, or have access to, a service like Amazon Web Service (AWS)[4]. Hecuba may initialize a node through AWS's API if the number of available nodes is zero, and the scaling-algorithm results in the need of extra nodes to the cluster. It is important to remember that such extension needs to be tested very carefully before it is applied to an implementation that is currently in use, as it may potentially cost a lot of money if something goes wrong.

Intelligent node insertion

The current version of Hecuba only initializes the new nodes, and let Cassandra handle the insertion point within the cluster. Cassandra has built-in functionality that determines which token range that has the highest load, and share this token range evenly between the new- and existing node. Since Hecuba may potentially insert and remove a lot of nodes, the cluster may end up being unbalanced. An unbalanced cluster leads to certain nodes being responsible for a larger amount of the overall data set. If a node contains more data than the rest, it may potentially receive the majority of the requests made to the cluster. When one node receives the majority of the connections, and holds the majority of the data within the cluster, the whole idea and the advantages of distributed systems are lost.

To prevent this skewness, a smarter token generator should be implemented. Newly inserted nodes should continue to find hotspots within the

cluster, to prevent nodes from being overloaded. But instead of splitting the token-range evenly between the existing, and the new node, the amount of data available at the existing node should be evenly shared instead. As the data is most likely not spread out evenly throughout the token-range, this would in most cases lead to one of the nodes being responsible for a larger amount of the token-range. Even though one of the nodes will have a larger token-range than the other, the amount of data is evenly shared, meaning that they should potentially receive an even amount of requests. An important thing to remember is that the cluster may be unbalanced again as soon as new data is inserted.

Automatically detect if the cluster is "too unbalanced"

As mentioned for intelligent node insertion, the cluster may end up being unbalanced at some point. This will always happen, not just as a result of intelligent node insertion. When the cluster becomes unbalanced, the performance may potentially be lowered, as there may be a higher amount of traffic directed to certain nodes.

To solve this, Hecuba should implement a daemon that runs in the background, detecting whenever the cluster reaches a certain point of skewness. This may be achieved by monitoring the load balance of the nodes in the cluster, and if the load-difference between two nodes is high, the token-ranges should be recalculated in order to balance the cluster based on the data that are currently within the cluster. The daemon should not be triggered too often, as rebalancing a cluster is very resource-intensive. It is very important that the daemon also consider peak-hours to prevent balancing the cluster while it is being actively used during e.g., holidays.

Consider data-center and rack when scaling

Hecuba does not consider the geographical location of the nodes when deciding where to insert or remove nodes. Transferring data between nodes that is located in different continents will take longer, and be more vulnerable against data-loss than transferring data between nodes that is located in the same country. Almost everything online, located all over the globe, is accessible through everyone's web-browser nowadays. This has resulted in the geographical distances being blurred out. Even though the distances have been blurred out, they will always make some impact. Retrieving data from the other side of the globe takes a few more milliseconds than retrieving data from a server located in your own town. Taken into consideration that both servers have the same incoming, and outgoing, bandwidth.

To compensate with the large amount of data traffic generated by Cassandra while moving data, Hecuba may take into consideration the data-center, and possibly the rack where each node is located. By doing this, the implementation will be able to take advantage of initializing nodes that is physically located closer to each other. Since the nodes are physically

located close, the data will be transferred across a shorter geographical distance, which will decrease the amount of time it takes to completely transfer all the data. It will also eliminate a lot of possible failures, as there are fewer external sources between the origin and the destination.

The data-center and rack information is available through Cassandra's nodetool utility by executing the `info` command. Hecuba should implement a way to retrieve this information for each node, and include this when scaling up or down nodes. At least it should consider the geographical location when it scales up. This is achieved by scanning through the list of available nodes, and selects the node that is closest to the node it is going to share token-range with. This would of course require the implementation to be extended with such meta-data for each node.

Initialization of agents should return a list of currently active nodes

The agent implementation does not include a list of currently active nodes in the initialization response to the master. The master will have to ask the agent for a list after it has been initiated. Including this list in the initial response from the agent, and rewrite the master-implementation to handle the received list should not be a lot of work. This was not done because there was not much time left for the thesis work when it was detected, and chaining this would delay the thesis since the tests should have been performed again. Even though we did not manage to implement this, it should be prioritized in a future release in order to achieve a fully automatic implementation.

When the agent receives the initialization-message from the master, it has to startup Cassandra if it is not already running. Furthermore it will have to ask Cassandra for a list of currently available nodes. The list may not be complete at once, which may require the implementation to sleep a second or so in order to make sure Cassandra updated its local list of nodes. When the list is retrieved, it will have to be put into the attributes-map of the message sent back to the master. The master listen for messages sent from the agents, and will read the response as soon as it arrives. The master should read the list of currently active nodes from the attribute-map. If there is a node in the list that was not already in the masters' local list of active nodes, an initialization-message will be sent in order to start monitoring these nodes as well. The master should also ask a random node for an updated list of currently active nodes at a regular basis, in order to update the local list, in case new nodes are added or removed manually.

By rewriting the implementation to handle this automatically, the autoscale implementation will be able to initialize, detect all nodes, and start working as intended without any supervision or input from database administrators.

Prevent the master from sending configuration-attributes if default constructor is used

The implementation appends the configuration attributes from the master to every new node that is initialized, except the initial node, which does not follow the normal initialization-flow. In a future release, the master should exclude the configuration attributes if the default constructor is used. The intension of the default constructor is that every agent may be configured differently through a local configuration-file. However, if there is included any configuration attributes to the initialization message sent by the master, the agent's local configurations will be overwritten. Therefore, the master implementation should be changed in order to exclude the configuration attributes if the default constructor is used.

The configuration attributes is included into the message by the `CassandraHostManager.addHostToCluster()` method. This method should be able to determine if the master implementation was initiated by the default constructor or not, and include or exclude the configuration attributes accordingly.

Stop monitoring when the node is scaled down

The node should stop monitoring when the node is scaled down. Today the nodes continue to monitor the local resource usage, and send breach-messages to the master if breaches occur after the node has been scaled down.

Whenever the agent receives a `STOP_AGENT` message from the master, the agent should decommission the node, kill the Cassandra process, wipe the local data and temporarily stop the resource monitoring. It should continue to listen for incoming messages, since the master may re-initialize the node later on. Currently, the only thing that is missing is to prevent the agent from monitoring its resources when the node is scaled down. The master may register the node as active since it receives breach-messages, which may result in the master trying to scale the node either up or down, which may result in inconsistency when it comes to the master's local lists of active-, and inactive-nodes.

Prevent downscaling to occur during peak-hours

Hecuba does not consider if there is heavy traffic or not when scaling down. This may be an issue if a downscaling is initiated while a lot of users are accessing the database. A downscale will occupy a lot of the local resources, which may prevent other processes from finishing e.g., users trying to retrieve data. This may result in poor user-experience for the users that are accessing the database as it may take longer time to retrieve data. Since Cassandra is built in order to provide good performance, the implementation should be further developed in order to prevent downscaling during peak hours.

Downscaling should not be triggered during periods when there are

possibilities that the cluster will be actively used, e.g., during holidays or bigger events. If the cluster will be used during a football match, it should never perform a scale down while the match is being played, in order to prevent users from experience longer response-time, and maybe even bring down the whole cluster.

Bibliography

[1] The netflix tech blog: Announcing priam. http://techblog.netflix.com/2012/02/announcing-priam.html. [Online; accessed 20-February-2013].

[2] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. A view of cloud computing. *Commun. ACM*, 53(4):50–58, April 2010.

[3] The netflix tech blog: Auto scaling in the amazon cloud. http://techblog.netflix.com/2012/01/auto-scaling-in-amazon-cloud.html. [Online; accessed 20-February-2013].

[4] Amazon web services, cloud computing: Compute, storage, database. http://aws.amazon.com/. [Online; accessed 20-February-2013].

[5] Daniel Bartholomew. Sql vs. nosql. *Linux J.*, 2010, July 2010.

[6] Jim Benson. *Personal kanban; mapping work, navigating life*. Modus Cooperandi Press, Seattle, WA, 2011.

[7] Cap theorem - wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/CAP_theorem. [Online; accessed 07-february-2013].

[8] Cassandra - wikipedia, the free encyclopaedia. http://en.wikipedia.org/wiki/Cassandra. [Online; accessed 06-February-2013].

[9] Rick Cattell. Scalable sql and nosql data stores. *SIGMOD Rec.*, 39:12–27, May 2011.

[10] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber. Bigtable: A distributed storage system for structured data. *ACM Trans. Comput. Syst.*, 26:4:1–4:26, June 2008.

[11] The netflix tech blog: Chaos monkey released into the wild. http://techblog.netflix.com/2012/07/chaos-monkey-released-into-wild.html. [Online; accessed 20-February-2013].

[12] Datamodel - cassandra wiki. http://wiki.apache.org/cassandra/DataModel. [Online; accessed 27-August-2012].

[13] Datastax opscenter : Datastax. http://www.datastax.com/products/opscenter. [Online; accessed 27-March-2013].

[14] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. Dynamo: amazon's highly available key-value store. *SIGOPS Oper. Syst. Rev.*, 41:205–220, October 2007.

[15] Twitter engineering: Introducing flockdb. http://engineering.twitter.com/2010/05/introducing-flockdb.html. [Online; accessed 14-February-2013].

[16] Git - official website. http://git-scm.com/. [Online; accessed 18-October-2012].

[17] Github. https://github.com/. [Online; accessed 18-October-2012].

[18] Hector - java client for cassandra. http://hector-client.github.com/hector/build/html/index.html. [Online; accessed 17-September-2012].

[19] Hecuba - wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/Hecuba. [Online; accessed 14-December-2012].

[20] Eben Hewitt. *Cassandra: The Definitive Guide*. O'Reilly Media, Inc., 1 edition, 2010.

[21] Eben Hewitt. *Cassandra: The Definitive Guide*, chapter Appendix, pages 17–19. O'Reilly Media, Inc., 1 edition, 2010.

[22] Sigar api (system information gatherer and reporter) | hyperic. http://www.hyperic.com/products/sigar. [Online; accessed 06-March-2013].

[23] Avinash Lakshman and Prashant Malik. Cassandra: a decentralized structured storage system. *SIGOPS Oper. Syst. Rev.*, 44:35–40, April 2010.

[24] Linode - xen vps hosting. [Online; accessed 14-March-2013].

[25] Linus thorvalds - wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/Linus_Torvalds. [Online; accessed 26-March-2013].

[26] Mike Loukides. The nosql movement - o'reilly radar. http://radar.oreilly.com/2012/02/nosql-non-relational-database.html. [Online; accessed 13-February-2012].

[27] Maven - welcome to apache maven. http://maven.apache.org/. [Online; accessed 04-March-2013].

[28] Learn, develop, participate - neo4j: The world's leading graph database. http://www.neo4j.org/. [Online; accessed 14-February-2013].

[29] Benchmarking cassandra scalability on aws - over a million writes per second. http://techblog.netflix.com/2011/11/benchmarking-cassandra-scalability-on.html. [Online; accessed 13-December.2012].

[30] The apache cassandra project. http://cassandra.apache.org/. [Online; accessed 02-January-2013].

[31] Cassandra wiki. http://wiki.apache.org/cassandra/. [Online; accessed 11-March-2012].

[32] Netflix/priam - github. https://github.com/Netflix/Priam/wiki. [Online; accessed 14-December-2012].

[33] Priam - wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/Priam. [Online; accessed 02-January-2013].

[34] Scrum (development) - wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/Scrum_(development). [Online; accessed 14-December-2012].

[35] Springsource.org. http://www.springsource.org/. [Online; accessed 27-February-2013].

[36] Spring bean definition. http://www.tutorialspoint.com/spring/spring_bean_definition.htm. [Online; accessed 22-April-2013].

[37] What is a story point ? | agilefaq. http://agilefaq.wordpress.com/2007/11/13/what-is-a-story-point/. [Online; accessed 22-April-2013].

[38] Ran Tavory. Understanding cassandra code base | prettyprint.me. http://prettyprint.me/2010/05/02/understanding-cassandra-code-base/. [Online; accessed 11-March-2012].

Appendices

TC1_H, Node A

TIMESTAMP	DISK USAGE	MEMORY USAGE	CPU USAGE
2013-01-24 15:44:21	72K	553708 kB	0.07%
2013-01-24 15:44:27	72K	563956 kB	0.07%
2013-01-24 15:44:32	72K	564848 kB	0.07%
2013-01-24 15:44:37	72K	564964 kB	0.07%
2013-01-24 15:44:42	72K	565220 kB	0.07%
2013-01-24 15:44:47	72K	565320 kB	0.07%
2013-01-24 15:44:52	72K	565452 kB	0.07%
2013-01-24 15:44:57	72K	565428 kB	0.07%
2013-01-24 15:45:02	72K	565560 kB	0.07%
2013-01-24 15:45:07	72K	565436 kB	0.07%
2013-01-24 15:45:12	72K	565816 kB	0.07%
2013-01-24 15:45:17	72K	565940 kB	0.07%
2013-01-24 15:45:22	72K	566676 kB	0.07%
2013-01-24 15:45:27	72K	566792 kB	0.07%
2013-01-24 15:45:32	72K	566940 kB	0.07%
2013-01-24 15:45:37	72K	567064 kB	0.07%
2013-01-24 15:45:42	72K	567188 kB	0.07%
2013-01-24 15:45:47	72K	567056 kB	0.07%
2013-01-24 15:45:52	72K	567172 kB	0.07%
2013-01-24 15:45:57	72K	567684 kB	0.07%
2013-01-24 15:46:02	72K	567676 kB	0.07%
2013-01-24 15:46:07	72K	567792 kB	0.07%
2013-01-24 15:46:12	72K	567924 kB	0.07%
2013-01-24 15:46:17	72K	568164 kB	0.07%
2013-01-24 15:46:22	72K	568164 kB	0.07%
2013-01-24 15:46:27	72K	568288 kB	0.07%
2013-01-24 15:46:32	72K	568420 kB	0.07%
2013-01-24 15:46:37	72K	568668 kB	0.07%
2013-01-24 15:46:42	72K	568652 kB	0.07%
2013-01-24 15:46:47	72K	568908 kB	0.07%
2013-01-24 15:46:53	72K	569040 kB	0.07%
2013-01-24 15:46:58	72K	569172 kB	0.07%
2013-01-24 15:47:03	72K	569296 kB	0.07%
2013-01-24 15:47:08	72K	569156 kB	0.07%
2013-01-24 15:47:13	72K	569568 kB	0.07%
2013-01-24 15:47:18	72K	569444 kB	0.07%
2013-01-24 15:47:23	72K	569816 kB	0.07%
2013-01-24 15:47:28	72K	569676 kB	0.07%
2013-01-24 15:47:33	72K	569816 kB	0.07%
2013-01-24 15:47:38	72K	569948 kB	0.07%
2013-01-24 15:47:43	72K	570048 kB	0.07%
2013-01-24 15:47:48	72K	570196 kB	0.07%
2013-01-24 15:47:53	72K	570196 kB	0.07%
2013-01-24 15:47:58	72K	570312 kB	0.07%
2013-01-24 15:48:03	72K	570428 kB	0.07%
2013-01-24 15:48:08	72K	570684 kB	0.07%
2013-01-24 15:48:13	72K	571156 kB	0.07%
2013-01-24 15:48:18	72K	571000 kB	0.07%
2013-01-24 15:48:23	72K	571016 kB	0.07%
2013-01-24 15:48:28	72K	571272 kB	0.07%
2013-01-24 15:48:33	72K	571388 kB	0.07%
2013-01-24 15:48:38	72K	571644 kB	0.07%
2013-01-24 15:48:43	72K	571760 kB	0.07%
2013-01-24 15:48:48	72K	571644 kB	0.07%
2013-01-24 15:48:53	72K	571652 kB	0.07%
2013-01-24 15:48:58	72K	572008 kB	0.07%
2013-01-24 15:49:03	72K	571776 kB	0.07%
2013-01-24 15:49:08	72K	571884 kB	0.07%
2013-01-24 15:49:14	72K	571992 kB	0.07%
2013-01-24 15:49:19	72K	572132 kB	0.07%
2013-01-24 15:49:24	72K	572388 kB	0.07%
2013-01-24 15:49:29	72K	572512 kB	0.07%
2013-01-24 15:49:34	72K	572636 kB	0.07%
2013-01-24 15:49:39	72K	572728 kB	0.07%
2013-01-24 15:49:44	72K	573504 kB	0.07%
2013-01-24 15:49:49	72K	573496 kB	0.07%
2013-01-24 15:49:54	72K	573752 kB	0.07%
2013-01-24 15:49:59	72K	573876 kB	0.07%
2013-01-24 15:50:04	72K	574124 kB	0.07%
2013-01-24 15:50:09	72K	574240 kB	0.07%
2013-01-24 15:50:14	72K	574348 kB	0.07%
2013-01-24 15:50:19	72K	574472 kB	0.07%
2013-01-24 15:50:24	72K	574744 kB	0.07%
2013-01-24 15:50:29	72K	574860 kB	0.07%
2013-01-24 15:50:34	72K	575116 kB	0.07%
2013-01-24 15:50:39	72K	575224 kB	0.07%
2013-01-24 15:50:44	72K	575480 kB	0.07%
2013-01-24 15:50:49	72K	575976 kB	0.07%
2013-01-24 15:50:54	72K	575852 kB	0.07%
2013-01-24 15:50:59	72K	575984 kB	0.07%
2013-01-24 15:51:04	72K	576224 kB	0.07%
2013-01-24 15:51:09	72K	576216 kB	0.07%
2013-01-24 15:51:14	72K	576356 kB	0.07%
2013-01-24 15:51:19	72K	576332 kB	0.07%

2013-01-24 15:51:24	72K	576456 kB	0.07%
2013-01-24 15:51:30	72K	576588 kB	0.07%
2013-01-24 15:51:35	72K	576844 kB	0.07%
2013-01-24 15:51:40	72K	576976 kB	0.07%
2013-01-24 15:51:45	72K	577340 kB	0.07%
2013-01-24 15:51:50	72K	577324 kB	0.07%
2013-01-24 15:51:55	72K	577464 kB	0.07%
2013-01-24 15:52:00	72K	577712 kB	0.07%
2013-01-24 15:52:05	72K	577844 kB	0.07%
2013-01-24 15:52:10	72K	577844 kB	0.07%
2013-01-24 15:52:15	72K	578068 kB	0.07%
2013-01-24 15:52:20	72K	578084 kB	0.07%
2013-01-24 15:52:25	72K	578340 kB	0.07%
2013-01-24 15:52:30	72K	578472 kB	0.07%
2013-01-24 15:52:35	72K	578828 kB	0.07%
2013-01-24 15:52:40	72K	578688 kB	0.07%
2013-01-24 15:52:45	72K	579084 kB	0.07%
2013-01-24 15:52:50	72K	579084 kB	0.07%
2013-01-24 15:52:55	72K	579208 kB	0.07%
2013-01-24 15:53:00	72K	579192 kB	0.07%
2013-01-24 15:53:05	72K	579580 kB	0.07%
2013-01-24 15:53:10	72K	579828 kB	0.07%
2013-01-24 15:53:15	72K	579960 kB	0.07%
2013-01-24 15:53:20	72K	580208 kB	0.07%
2013-01-24 15:53:25	72K	580324 kB	0.07%
2013-01-24 15:53:30	72K	580448 kB	0.07%
2013-01-24 15:53:35	72K	580704 kB	0.07%
2013-01-24 15:53:40	72K	580952 kB	0.07%
2013-01-24 15:53:46	72K	581060 kB	0.07%
2013-01-24 15:53:51	72K	581192 kB	0.07%
2013-01-24 15:53:56	72K	581448 kB	0.07%
2013-01-24 15:54:01	72K	581200 kB	0.07%
2013-01-24 15:54:06	72K	581464 kB	0.07%
2013-01-24 15:54:11	72K	581556 kB	0.07%
2013-01-24 15:54:16	72K	581680 kB	0.07%
2013-01-24 15:54:21	72K	581936 kB	0.07%
2013-01-24 15:54:26	72K	582060 kB	0.07%
2013-01-24 15:54:31	72K	582316 kB	0.07%
2013-01-24 15:54:36	72K	582432 kB	0.07%
2013-01-24 15:54:41	72K	582556 kB	0.07%
2013-01-24 15:54:46	72K	582804 kB	0.07%
2013-01-24 15:54:51	72K	582664 kB	0.07%
2013-01-24 15:54:56	72K	583028 kB	0.07%
2013-01-24 15:55:01	72K	583176 kB	0.07%
2013-01-24 15:55:06	72K	583168 kB	0.07%
2013-01-24 15:55:11	72K	583424 kB	0.07%
2013-01-24 15:55:16	72K	583820 kB	0.07%
2013-01-24 15:55:21	72K	583952 kB	0.07%
2013-01-24 15:55:26	72K	584192 kB	0.07%
2013-01-24 15:55:31	72K	584200 kB	0.07%
2013-01-24 15:55:36	72K	584448 kB	0.07%
2013-01-24 15:55:41	72K	584804 kB	0.07%
2013-01-24 15:55:46	72K	584936 kB	0.07%
2013-01-24 15:55:51	72K	585052 kB	0.07%
2013-01-24 15:55:56	72K	585060 kB	0.07%
2013-01-24 15:56:01	72K	585168 kB	0.07%
2013-01-24 15:56:06	72K	585160 kB	0.07%
2013-01-24 15:56:12	72K	585424 kB	0.07%
2013-01-24 15:56:17	72K	585664 kB	0.07%
2013-01-24 15:56:22	72K	585896 kB	0.07%
2013-01-24 15:56:27	72K	585896 kB	0.07%
2013-01-24 15:56:32	72K	586028 kB	0.07%
2013-01-24 15:56:37	72K	586268 kB	0.07%
2013-01-24 15:56:42	72K	586268 kB	0.07%
2013-01-24 15:56:47	72K	586392 kB	0.07%
2013-01-24 15:56:52	72K	586640 kB	0.07%
2013-01-24 15:56:57	72K	586764 kB	0.07%
2013-01-24 15:57:02	72K	586904 kB	0.07%
2013-01-24 15:57:07	72K	587020 kB	0.07%
2013-01-24 15:57:12	72K	587144 kB	0.07%
2013-01-24 15:57:17	72K	587368 kB	0.07%
2013-01-24 15:57:22	72K	587508 kB	0.07%
2013-01-24 15:57:27	72K	587516 kB	0.07%
2013-01-24 15:57:32	72K	587764 kB	0.07%
2013-01-24 15:57:37	72K	587912 kB	0.07%
2013-01-24 15:57:42	72K	587872 kB	0.07%
2013-01-24 15:57:47	72K	587880 kB	0.07%
2013-01-24 15:57:52	72K	588136 kB	0.07%
2013-01-24 15:57:57	72K	588384 kB	0.07%
2013-01-24 15:58:02	72K	588508 kB	0.07%
2013-01-24 15:58:07	72K	588632 kB	0.07%
2013-01-24 15:58:12	72K	588880 kB	0.07%
2013-01-24 15:58:17	72K	588748 kB	0.07%
2013-01-24 15:58:22	72K	589012 kB	0.07%
2013-01-24 15:58:27	72K	589368 kB	0.07%
2013-01-24 15:58:33	72K	589500 kB	0.07%
2013-01-24 15:58:38	72K	589624 kB	0.07%
2013-01-24 15:58:43	72K	589880 kB	0.07%
2013-01-24 15:58:48	72K	590120 kB	0.07%
2013-01-24 15:58:53	72K	590220 kB	0.07%
2013-01-24 15:58:58	72K	590080 kB	0.07%
2013-01-24 15:59:03	72K	590120 kB	0.07%
2013-01-24 15:59:08	72K	590236 kB	0.07%
2013-01-24 15:59:13	72K	590492 kB	0.07%
2013-01-24 15:59:18	72K	590600 kB	0.07%
2013-01-24 15:59:23	72K	590864 kB	0.07%
2013-01-24 15:59:28	72K	591112 kB	0.07%

TC1_H, Node B

TIMESTAMP	DISK USAGE	MEMORY USAGE	CPU USAGE
2013-01-24 11:10:24	72K	770388 kB	19.55%
2013-01-24 11:10:29	72K	782208 kB	19.55%
2013-01-24 11:10:34	72K	782820 kB	19.55%
2013-01-24 11:10:39	72K	783208 kB	19.55%
2013-01-24 11:10:44	72K	783456 kB	19.55%
2013-01-24 11:10:49	72K	783952 kB	19.55%
2013-01-24 11:10:54	72K	783936 kB	19.55%
2013-01-24 11:10:59	72K	783952 kB	19.55%
2013-01-24 11:11:04	72K	783984 kB	19.55%
2013-01-24 11:11:09	72K	784380 kB	19.55%
2013-01-24 11:11:14	72K	784224 kB	19.55%
2013-01-24 11:11:19	72K	784480 kB	19.55%
2013-01-24 11:11:24	72K	784728 kB	19.55%
2013-01-24 11:11:29	72K	784728 kB	19.55%
2013-01-24 11:11:34	72K	784952 kB	19.55%
2013-01-24 11:11:39	72K	785076 kB	19.55%
2013-01-24 11:11:44	72K	784952 kB	19.55%
2013-01-24 11:11:49	72K	785324 kB	19.55%
2013-01-24 11:11:54	72K	785564 kB	19.55%
2013-01-24 11:11:59	72K	785456 kB	19.55%
2013-01-24 11:12:04	72K	786116 kB	19.55%
2013-01-24 11:12:09	72K	786116 kB	19.55%
2013-01-24 11:12:14	72K	786076 kB	19.55%
2013-01-24 11:12:19	72K	786448 kB	19.55%
2013-01-24 11:12:24	72K	786332 kB	19.55%
2013-01-24 11:12:29	72K	786580 kB	19.55%
2013-01-24 11:12:34	72K	786548 kB	19.55%
2013-01-24 11:12:39	72K	786432 kB	19.55%
2013-01-24 11:12:44	72K	786432 kB	19.55%
2013-01-24 11:12:49	72K	787068 kB	19.55%
2013-01-24 11:12:54	72K	786912 kB	19.55%
2013-01-24 11:12:59	72K	787308 kB	19.55%
2013-01-24 11:13:04	72K	787936 kB	19.55%
2013-01-24 11:13:09	72K	788076 kB	19.55%
2013-01-24 11:13:14	72K	788200 kB	19.55%
2013-01-24 11:13:19	72K	788324 kB	19.55%
2013-01-24 11:13:24	72K	788572 kB	19.55%
2013-01-24 11:13:29	72K	788720 kB	19.55%
2013-01-24 11:13:34	72K	789084 kB	19.55%
2013-01-24 11:13:39	72K	788960 kB	19.55%
2013-01-24 11:13:44	72K	788712 kB	19.55%
2013-01-24 11:13:49	72K	789332 kB	19.55%
2013-01-24 11:13:54	72K	789424 kB	19.55%
2013-01-24 11:13:59	72K	789688 kB	19.55%
2013-01-24 11:14:05	72K	789860 kB	19.55%
2013-01-24 11:14:10	72K	789852 kB	19.55%
2013-01-24 11:14:15	72K	789928 kB	19.55%
2013-01-24 11:14:20	72K	790052 kB	19.55%
2013-01-24 11:14:25	72K	790192 kB	19.55%
2013-01-24 11:14:30	72K	790564 kB	19.55%
2013-01-24 11:14:35	72K	790656 kB	19.55%
2013-01-24 11:14:40	72K	790804 kB	19.55%
2013-01-24 11:14:45	72K	791044 kB	19.55%
2013-01-24 11:14:50	72K	791060 kB	19.55%
2013-01-24 11:14:55	72K	791292 kB	19.55%
2013-01-24 11:15:00	72K	791540 kB	19.55%
2013-01-24 11:15:05	72K	791920 kB	19.55%
2013-01-24 11:15:10	72K	792044 kB	19.55%
2013-01-24 11:15:15	72K	791888 kB	19.55%
2013-01-24 11:15:20	72K	792036 kB	19.55%
2013-01-24 11:15:25	72K	792168 kB	19.55%
2013-01-24 11:15:30	72K	792168 kB	19.55%
2013-01-24 11:15:35	72K	792384 kB	19.55%
2013-01-24 11:15:40	72K	792756 kB	19.55%
2013-01-24 11:15:45	72K	792756 kB	19.55%
2013-01-24 11:15:50	72K	792880 kB	19.55%
2013-01-24 11:15:55	72K	792896 kB	19.55%
2013-01-24 11:16:00	72K	793044 kB	19.55%
2013-01-24 11:16:05	72K	793680 kB	19.55%
2013-01-24 11:16:10	72K	793516 kB	19.55%
2013-01-24 11:16:15	72K	793756 kB	19.55%
2013-01-24 11:16:20	72K	793756 kB	19.55%
2013-01-24 11:16:25	72K	793772 kB	19.55%
2013-01-24 11:16:30	72K	793996 kB	19.55%
2013-01-24 11:16:35	72K	794012 kB	19.55%
2013-01-24 11:16:40	72K	794144 kB	19.55%
2013-01-24 11:16:45	72K	794268 kB	19.55%
2013-01-24 11:16:50	72K	794608 kB	19.55%
2013-01-24 11:16:55	72K	794872 kB	19.55%
2013-01-24 11:17:00	72K	795376 kB	19.55%
2013-01-24 11:17:05	72K	795500 kB	19.55%
2013-01-24 11:17:10	72K	795344 kB	19.55%
2013-01-24 11:17:15	72K	795476 kB	19.55%
2013-01-24 11:17:20	72K	796112 kB	19.55%
2013-01-24 11:17:25	72K	796252 kB	19.55%
2013-01-24 11:17:30	72K	796244 kB	19.55%

2013-01-24 11:17:35	72K	796368 kB	19.55%
2013-01-24 11:17:40	72K	796740 kB	19.55%
2013-01-24 11:17:45	72K	796748 kB	19.55%
2013-01-24 11:17:50	72K	796988 kB	19.55%
2013-01-24 11:17:55	72K	796740 kB	19.55%
2013-01-24 11:18:00	72K	796996 kB	19.55%
2013-01-24 11:18:05	72K	797516 kB	19.55%
2013-01-24 11:18:10	72K	797352 kB	19.55%
2013-01-24 11:18:15	72K	797600 kB	19.55%
2013-01-24 11:18:20	72K	797600 kB	19.55%
2013-01-24 11:18:26	72K	797600 kB	19.55%
2013-01-24 11:18:31	72K	797700 kB	19.55%
2013-01-24 11:18:36	72K	797716 kB	19.55%
2013-01-24 11:18:41	72K	798492 kB	19.55%
2013-01-24 11:18:46	72K	798616 kB	19.55%
2013-01-24 11:18:51	72K	798832 kB	19.55%
2013-01-24 11:18:56	72K	799096 kB	19.55%
2013-01-24 11:19:01	72K	798972 kB	19.55%
2013-01-24 11:19:06	72K	799344 kB	19.55%
2013-01-24 11:19:11	72K	799312 kB	19.55%
2013-01-24 11:19:16	72K	799452 kB	19.55%
2013-01-24 11:19:21	72K	799460 kB	19.55%
2013-01-24 11:19:26	72K	799708 kB	19.55%
2013-01-24 11:19:31	72K	800312 kB	19.55%
2013-01-24 11:19:36	72K	800336 kB	19.55%
2013-01-24 11:19:41	72K	800460 kB	19.54%
2013-01-24 11:19:46	72K	801104 kB	19.54%
2013-01-24 11:19:51	72K	801344 kB	19.54%
2013-01-24 11:19:56	72K	801344 kB	19.54%
2013-01-24 11:20:01	72K	801352 kB	19.54%
2013-01-24 11:20:06	72K	801236 kB	19.54%
2013-01-24 11:20:11	72K	801444 kB	19.54%
2013-01-24 11:20:16	72K	801560 kB	19.54%
2013-01-24 11:20:21	72K	801560 kB	19.54%
2013-01-24 11:20:26	72K	801808 kB	19.54%
2013-01-24 11:20:31	72K	801924 kB	19.54%
2013-01-24 11:20:36	72K	801940 kB	19.54%
2013-01-24 11:20:41	72K	802460 kB	19.54%
2013-01-24 11:20:46	72K	802832 kB	19.54%
2013-01-24 11:20:51	72K	802800 kB	19.54%
2013-01-24 11:20:56	72K	803296 kB	19.54%
2013-01-24 11:21:01	72K	803296 kB	19.54%
2013-01-24 11:21:06	72K	803180 kB	19.54%
2013-01-24 11:21:11	72K	803304 kB	19.54%
2013-01-24 11:21:16	72K	803304 kB	19.54%
2013-01-24 11:21:21	72K	803428 kB	19.54%
2013-01-24 11:21:26	72K	803924 kB	19.54%
2013-01-24 11:21:31	72K	803784 kB	19.54%
2013-01-24 11:21:36	72K	804056 kB	19.54%
2013-01-24 11:21:41	72K	804304 kB	19.54%
2013-01-24 11:21:46	72K	804528 kB	19.54%
2013-01-24 11:21:51	72K	804660 kB	19.54%
2013-01-24 11:21:56	72K	804668 kB	19.54%
2013-01-24 11:22:01	72K	805304 kB	19.54%
2013-01-24 11:22:06	72K	805196 kB	19.54%
2013-01-24 11:22:11	72K	805064 kB	19.54%
2013-01-24 11:22:16	72K	805444 kB	19.54%
2013-01-24 11:22:21	72K	805940 kB	19.54%
2013-01-24 11:22:26	72K	805660 kB	19.54%
2013-01-24 11:22:31	72K	805792 kB	19.54%
2013-01-24 11:22:37	72K	805916 kB	19.54%
2013-01-24 11:22:42	72K	805916 kB	19.54%
2013-01-24 11:22:47	72K	806388 kB	19.54%
2013-01-24 11:22:52	72K	806388 kB	19.54%
2013-01-24 11:22:57	72K	806636 kB	19.54%
2013-01-24 11:23:02	72K	806760 kB	19.54%
2013-01-24 11:23:07	72K	806884 kB	19.54%
2013-01-24 11:23:12	72K	807412 kB	19.54%
2013-01-24 11:23:17	72K	807056 kB	19.54%
2013-01-24 11:23:22	72K	807304 kB	19.54%
2013-01-24 11:23:27	72K	807768 kB	19.54%
2013-01-24 11:23:32	72K	807396 kB	19.54%
2013-01-24 11:23:37	72K	807784 kB	19.54%
2013-01-24 11:23:42	72K	807916 kB	19.54%
2013-01-24 11:23:47	72K	809760 kB	19.54%
2013-01-24 11:23:52	72K	808520 kB	19.54%
2013-01-24 11:23:57	72K	808520 kB	19.54%
2013-01-24 11:24:02	72K	809412 kB	19.54%
2013-01-24 11:24:07	72K	809140 kB	19.54%
2013-01-24 11:24:12	72K	809272 kB	19.54%
2013-01-24 11:24:17	72K	809388 kB	19.54%
2013-01-24 11:24:22	72K	809760 kB	19.54%
2013-01-24 11:24:27	72K	809628 kB	19.54%
2013-01-24 11:24:32	72K	809768 kB	19.54%
2013-01-24 11:24:37	72K	810016 kB	19.54%
2013-01-24 11:24:42	72K	810140 kB	19.54%
2013-01-24 11:24:47	72K	810232 kB	19.54%
2013-01-24 11:24:52	72K	810496 kB	19.54%
2013-01-24 11:24:57	72K	810868 kB	19.54%
2013-01-24 11:25:02	72K	810868 kB	19.54%
2013-01-24 11:25:07	72K	811216 kB	19.54%
2013-01-24 11:25:12	72K	811092 kB	19.54%
2013-01-24 11:25:17	72K	811744 kB	19.54%
2013-01-24 11:25:22	72K	811644 kB	19.54%
2013-01-24 11:25:27	72K	811612 kB	19.54%

TC2, Node A

TIMESTAMP	DISK USAGE	MEMORY USAGE	CPU USAGE
2013-01-25 09:44:06	144K	545636 kB	0.07%
2013-01-25 09:44:11	144K	545900 kB	0.07%
2013-01-25 09:44:16	144K	546412 kB	0.07%
2013-01-25 09:44:21	144K	546008 kB	0.07%
2013-01-25 09:44:26	144K	546404 kB	0.07%
2013-01-25 09:44:31	144K	546280 kB	0.07%
2013-01-25 09:44:36	144K	546256 kB	0.07%
2013-01-25 09:44:41	144K	546528 kB	0.07%
2013-01-25 09:44:46	144K	546776 kB	0.07%
2013-01-25 09:44:51	144K	546768 kB	0.07%
2013-01-25 09:44:56	144K	546776 kB	0.07%
2013-01-25 09:45:01	144K	546892 kB	0.07%
2013-01-25 09:45:06	144K	547148 kB	0.07%
2013-01-25 09:45:11	144K	547016 kB	0.07%
2013-01-25 09:45:16	144K	547024 kB	0.07%
2013-01-25 09:45:21	144K	547412 kB	0.07%
2013-01-25 09:45:27	144K	547536 kB	0.07%
2013-01-25 09:45:32	144K	547520 kB	0.07%
2013-01-25 09:45:37	144K	547768 kB	0.07%
2013-01-25 09:45:42	144K	547652 kB	0.07%
2013-01-25 09:45:47	144K	547768 kB	0.07%
2013-01-25 09:45:52	144K	547776 kB	0.07%
2013-01-25 09:45:57	144K	547900 kB	0.07%
2013-01-25 09:46:02	144K	548024 kB	0.07%
2013-01-25 09:46:07	144K	548148 kB	0.07%
2013-01-25 09:46:12	144K	548008 kB	0.07%
2013-01-25 09:46:17	144K	548264 kB	0.07%
2013-01-25 09:46:22	144K	548264 kB	0.07%
2013-01-25 09:46:27	144K	548388 kB	0.07%
2013-01-25 09:46:32	144K	548512 kB	0.07%
2013-01-25 09:46:37	144K	548636 kB	0.07%
2013-01-25 09:46:42	144K	548636 kB	0.07%
2013-01-25 09:46:47	144K	548860 kB	0.07%
2013-01-25 09:46:52	144K	549124 kB	0.07%
2013-01-25 09:46:57	144K	549116 kB	0.07%
2013-01-25 09:47:02	144K	549232 kB	0.07%
2013-01-25 09:47:07	144K	549464 kB	0.07%
2013-01-25 09:47:12	144K	549660 kB	0.07%
2013-01-25 09:47:17	144K	549908 kB	0.07%
2013-01-25 09:47:22	144K	549644 kB	0.07%
2013-01-25 09:47:27	144K	549636 kB	0.07%
2013-01-25 09:47:32	144K	549884 kB	0.07%
2013-01-25 09:47:37	144K	550148 kB	0.07%
2013-01-25 09:47:42	144K	550404 kB	0.07%
2013-01-25 09:47:47	144K	550396 kB	0.07%
2013-01-25 09:47:53	144K	550784 kB	0.07%
2013-01-25 09:47:58	144K	550908 kB	0.07%
2013-01-25 09:48:03	144K	551024 kB	0.07%
2013-01-25 09:48:08	144K	551272 kB	0.07%
2013-01-25 09:48:13	144K	551264 kB	0.07%
2013-01-25 09:48:18	144K	551224 kB	0.07%
2013-01-25 09:48:23	144K	551472 kB	0.07%
2013-01-25 09:48:28	144K	551480 kB	0.07%
2013-01-25 09:48:33	144K	551728 kB	0.07%
2013-01-25 09:48:38	144K	551860 kB	0.07%
2013-01-25 09:48:43	144K	551984 kB	0.07%
2013-01-25 09:48:48	144K	551984 kB	0.07%
2013-01-25 09:48:53	144K	552108 kB	0.07%
2013-01-25 09:48:58	144K	552100 kB	0.07%
2013-01-25 09:49:03	144K	552340 kB	0.07%
2013-01-25 09:49:08	144K	552356 kB	0.07%
2013-01-25 09:49:13	144K	552488 kB	0.07%
2013-01-25 09:49:18	144K	552852 kB	0.07%
2013-01-25 09:49:23	144K	552828 kB	0.07%
2013-01-25 09:49:28	144K	552936 kB	0.07%
2013-01-25 09:49:33	144K	552968 kB	0.07%
2013-01-25 09:49:38	144K	553224 kB	0.07%
2013-01-25 09:49:43	144K	553348 kB	0.07%
2013-01-25 09:49:48	144K	553480 kB	0.07%
2013-01-25 09:49:53	144K	553456 kB	0.07%
2013-01-25 09:49:58	144K	553596 kB	0.07%
2013-01-25 09:50:03	144K	553712 kB	0.07%
2013-01-25 09:50:08	144K	553844 kB	0.07%
2013-01-25 09:50:13	144K	554232 kB	0.07%
2013-01-25 09:50:19	144K	554240 kB	0.07%
2013-01-25 09:50:24	144K	554488 kB	0.07%
2013-01-25 09:50:29	144K	554588 kB	0.07%
2013-01-25 09:50:34	144K	554728 kB	0.07%
2013-01-25 09:50:39	144K	554588 kB	0.07%
2013-01-25 09:50:44	144K	554588 kB	0.07%
2013-01-25 09:50:49	144K	554984 kB	0.07%
2013-01-25 09:50:54	144K	554596 kB	0.07%
2013-01-25 09:50:59	144K	554588 kB	0.07%
2013-01-25 09:51:04	144K	554860 kB	0.07%
2013-01-25 09:51:09	144K	554836 kB	0.07%
2013-01-25 09:51:14	144K	554952 kB	0.07%

2013-01-25 09:51:19	144K	554960 kB	0.07%
2013-01-25 09:51:24	144K	555084 kB	0.07%
2013-01-25 09:51:29	144K	555332 kB	0.07%
2013-01-25 09:51:34	144K	555456 kB	0.07%
2013-01-25 09:51:39	144K	555580 kB	0.07%
2013-01-25 09:51:44	144K	555440 kB	0.07%
2013-01-25 09:51:49	144K	555696 kB	0.07%
2013-01-25 09:51:54	144K	555696 kB	0.07%
2013-01-25 09:51:59	144K	555812 kB	0.07%
2013-01-25 09:52:04	144K	555828 kB	0.07%
2013-01-25 09:52:09	144K	555952 kB	0.07%
2013-01-25 09:52:14	144K	556208 kB	0.07%
2013-01-25 09:52:19	144K	556324 kB	0.07%
2013-01-25 09:52:24	144K	556448 kB	0.07%
2013-01-25 09:52:29	144K	556580 kB	0.07%
2013-01-25 09:52:34	144K	556696 kB	0.07%
2013-01-25 09:52:39	144K	556424 kB	0.07%
2013-01-25 09:52:44	144K	556820 kB	0.07%
2013-01-25 09:52:50	144K	556812 kB	0.07%
2013-01-25 09:52:55	144K	557192 kB	0.07%
2013-01-25 09:53:00	144K	557440 kB	0.07%
2013-01-25 09:53:05	144K	557176 kB	0.07%
2013-01-25 09:53:10	144K	557152 kB	0.07%
2013-01-25 09:53:15	144K	557432 kB	0.07%
2013-01-25 09:53:20	144K	557688 kB	0.07%
2013-01-25 09:53:25	144K	557688 kB	0.07%
2013-01-25 09:53:30	144K	557928 kB	0.07%
2013-01-25 09:53:35	144K	557928 kB	0.07%
2013-01-25 09:53:40	144K	557920 kB	0.07%
2013-01-25 09:53:45	144K	558184 kB	0.07%
2013-01-25 09:53:50	144K	558432 kB	0.07%
2013-01-25 09:53:55	144K	558556 kB	0.07%
2013-01-25 09:54:00	144K	558936 kB	0.07%
2013-01-25 09:54:05	144K	558524 kB	0.07%
2013-01-25 09:54:10	144K	558796 kB	0.07%
2013-01-25 09:54:15	144K	559052 kB	0.07%
2013-01-25 09:54:20	144K	559160 kB	0.07%
2013-01-25 09:54:25	144K	559424 kB	0.07%
2013-01-25 09:54:30	144K	559540 kB	0.07%
2013-01-25 09:54:35	144K	559672 kB	0.07%
2013-01-25 09:54:40	144K	559920 kB	0.07%
2013-01-25 09:54:45	144K	559912 kB	0.07%
2013-01-25 09:54:50	144K	560036 kB	0.07%
2013-01-25 09:54:55	144K	560168 kB	0.07%
2013-01-25 09:55:00	144K	560416 kB	0.07%
2013-01-25 09:55:05	144K	560664 kB	0.07%
2013-01-25 09:55:10	144K	560540 kB	0.07%
2013-01-25 09:55:16	144K	560664 kB	0.07%
2013-01-25 09:55:21	144K	560920 kB	0.07%
2013-01-25 09:55:26	144K	561168 kB	0.07%
2013-01-25 09:55:31	144K	561424 kB	0.07%
2013-01-25 09:55:36	144K	561424 kB	0.07%
2013-01-25 09:55:41	144K	561920 kB	0.07%
2013-01-25 09:55:46	144K	561748 kB	0.07%
2013-01-25 09:55:51	144K	562036 kB	0.07%
2013-01-25 09:55:56	144K	562160 kB	0.07%
2013-01-25 09:56:01	144K	562160 kB	0.07%
2013-01-25 09:56:06	144K	562268 kB	0.07%
2013-01-25 09:56:11	144K	562392 kB	0.07%
2013-01-25 09:56:16	144K	562640 kB	0.07%
2013-01-25 09:56:21	144K	562788 kB	0.07%
2013-01-25 09:56:26	144K	562756 kB	0.07%
2013-01-25 09:56:31	144K	562888 kB	0.07%
2013-01-25 09:56:36	144K	563152 kB	0.07%
2013-01-25 09:56:41	144K	563268 kB	0.07%
2013-01-25 09:56:46	144K	563516 kB	0.07%
2013-01-25 09:56:51	144K	563764 kB	0.07%
2013-01-25 09:56:56	144K	563880 kB	0.07%
2013-01-25 09:57:01	144K	563864 kB	0.07%
2013-01-25 09:57:06	144K	564120 kB	0.07%
2013-01-25 09:57:11	144K	564260 kB	0.07%
2013-01-25 09:57:16	144K	564508 kB	0.07%
2013-01-25 09:57:21	144K	564748 kB	0.07%
2013-01-25 09:57:26	144K	564880 kB	0.07%
2013-01-25 09:57:31	144K	565120 kB	0.07%
2013-01-25 09:57:36	144K	565252 kB	0.07%
2013-01-25 09:57:42	144K	565384 kB	0.07%
2013-01-25 09:57:47	144K	565368 kB	0.07%
2013-01-25 09:57:52	144K	565748 kB	0.07%
2013-01-25 09:57:57	144K	565880 kB	0.07%
2013-01-25 09:58:02	144K	566004 kB	0.07%
2013-01-25 09:58:07	144K	565980 kB	0.07%
2013-01-25 09:58:12	144K	566120 kB	0.07%
2013-01-25 09:58:17	144K	566492 kB	0.07%
2013-01-25 09:58:22	144K	566632 kB	0.07%
2013-01-25 09:58:27	144K	566848 kB	0.07%
2013-01-25 09:58:32	144K	566864 kB	0.07%
2013-01-25 09:58:37	144K	567112 kB	0.07%
2013-01-25 09:58:42	144K	567220 kB	0.07%
2013-01-25 09:58:47	144K	567352 kB	0.07%
2013-01-25 09:58:52	144K	567244 kB	0.07%
2013-01-25 09:58:57	144K	567360 kB	0.07%
2013-01-25 09:59:02	144K	567732 kB	0.07%
2013-01-25 09:59:07	144K	567856 kB	0.07%
2013-01-25 09:59:12	144K	567848 kB	0.07%

TC2, Node B

TIMESTAMP	DISK USAGE	MEMORY USAGE	CPU USAGE
2013-01-25 05:10:07	144K	767688 kB	19.19%
2013-01-25 05:10:12	144K	768084 kB	19.19%
2013-01-25 05:10:17	144K	768456 kB	19.19%
2013-01-25 05:10:22	144K	768232 kB	19.19%
2013-01-25 05:10:27	144K	768480 kB	19.19%
2013-01-25 05:10:32	144K	768728 kB	19.19%
2013-01-25 05:10:37	144K	770060 kB	19.19%
2013-01-25 05:10:42	144K	769192 kB	19.19%
2013-01-25 05:10:47	144K	769812 kB	19.19%
2013-01-25 05:10:52	144K	769796 kB	19.19%
2013-01-25 05:10:57	144K	769672 kB	19.19%
2013-01-25 05:11:02	144K	769688 kB	19.19%
2013-01-25 05:11:07	144K	769836 kB	19.19%
2013-01-25 05:11:12	144K	770076 kB	19.19%
2013-01-25 05:11:17	144K	770200 kB	19.19%
2013-01-25 05:11:22	144K	769952 kB	19.19%
2013-01-25 05:11:27	144K	769828 kB	19.19%
2013-01-25 05:11:32	144K	770540 kB	19.19%
2013-01-25 05:11:37	144K	770696 kB	19.19%
2013-01-25 05:11:42	144K	770944 kB	19.19%
2013-01-25 05:11:47	144K	771316 kB	19.19%
2013-01-25 05:11:52	144K	771284 kB	19.19%
2013-01-25 05:11:57	144K	771672 kB	19.19%
2013-01-25 05:12:02	144K	771812 kB	19.19%
2013-01-25 05:12:07	144K	771812 kB	19.19%
2013-01-25 05:12:12	144K	771764 kB	19.19%
2013-01-25 05:12:17	144K	772044 kB	19.19%
2013-01-25 05:12:22	144K	772168 kB	19.19%
2013-01-25 05:12:27	144K	772548 kB	19.19%
2013-01-25 05:12:32	144K	772400 kB	19.19%
2013-01-25 05:12:37	144K	772780 kB	19.19%
2013-01-25 05:12:42	144K	772780 kB	19.19%
2013-01-25 05:12:47	144K	773292 kB	19.19%
2013-01-25 05:12:53	144K	773392 kB	19.19%
2013-01-25 05:12:58	144K	773400 kB	19.19%
2013-01-25 05:13:03	144K	773664 kB	19.19%
2013-01-25 05:13:08	144K	774036 kB	19.19%
2013-01-25 05:13:13	144K	774036 kB	19.19%
2013-01-25 05:13:18	144K	774168 kB	19.19%
2013-01-25 05:13:23	144K	774176 kB	19.19%
2013-01-25 05:13:28	144K	774556 kB	19.19%
2013-01-25 05:13:33	144K	774648 kB	19.19%
2013-01-25 05:13:38	144K	774920 kB	19.19%
2013-01-25 05:13:43	144K	774920 kB	19.19%
2013-01-25 05:13:48	144K	775176 kB	19.19%
2013-01-25 05:13:53	144K	775664 kB	19.19%
2013-01-25 05:13:58	144K	775548 kB	19.19%
2013-01-25 05:14:03	144K	776068 kB	19.19%
2013-01-25 05:14:08	144K	775688 kB	19.19%
2013-01-25 05:14:13	144K	776152 kB	19.19%
2013-01-25 05:14:18	144K	776308 kB	19.19%
2013-01-25 05:14:23	144K	776440 kB	19.19%
2013-01-25 05:14:28	144K	776160 kB	19.19%
2013-01-25 05:14:33	144K	776748 kB	19.19%
2013-01-25 05:14:38	144K	777492 kB	19.19%
2013-01-25 05:14:43	144K	777996 kB	19.19%
2013-01-25 05:14:48	144K	778344 kB	19.19%
2013-01-25 05:14:53	144K	778724 kB	19.19%
2013-01-25 05:14:58	144K	777128 kB	19.19%
2013-01-25 05:15:03	144K	777276 kB	19.19%
2013-01-25 05:15:08	144K	777500 kB	19.19%
2013-01-25 05:15:13	144K	778128 kB	19.19%
2013-01-25 05:15:18	144K	778260 kB	19.19%
2013-01-25 05:15:23	144K	778260 kB	19.19%
2013-01-25 05:15:28	144K	778228 kB	19.19%
2013-01-25 05:15:33	144K	778236 kB	19.19%
2013-01-25 05:15:38	144K	778980 kB	19.19%
2013-01-25 05:15:43	144K	778608 kB	19.19%
2013-01-25 05:15:48	144K	778840 kB	19.19%
2013-01-25 05:15:53	144K	779220 kB	19.19%
2013-01-25 05:15:58	144K	779616 kB	19.19%
2013-01-25 05:16:03	144K	779912 kB	19.19%
2013-01-25 05:16:08	144K	779872 kB	19.19%
2013-01-25 05:16:13	144K	779740 kB	19.19%
2013-01-25 05:16:18	144K	780260 kB	19.19%
2013-01-25 05:16:23	144K	780508 kB	19.19%
2013-01-25 05:16:28	144K	780600 kB	19.19%
2013-01-25 05:16:33	144K	780624 kB	19.19%
2013-01-25 05:16:38	144K	781244 kB	19.19%
2013-01-25 05:16:43	144K	780996 kB	19.19%
2013-01-25 05:16:48	144K	780964 kB	19.19%
2013-01-25 05:16:54	144K	781352 kB	19.19%
2013-01-25 05:16:59	144K	781252 kB	19.19%
2013-01-25 05:17:04	144K	781376 kB	19.19%
2013-01-25 05:17:09	144K	781716 kB	19.19%
2013-01-25 05:17:14	144K	781716 kB	19.19%

2013-01-25 05:17:19	144K	782104 kB	19.19%
2013-01-25 05:17:24	144K	782756 kB	19.19%
2013-01-25 05:17:29	144K	782352 kB	19.19%
2013-01-25 05:17:34	144K	782476 kB	19.19%
2013-01-25 05:17:39	144K	782600 kB	19.19%
2013-01-25 05:17:44	144K	782848 kB	19.19%
2013-01-25 05:17:49	144K	783072 kB	19.19%
2013-01-25 05:17:54	144K	783212 kB	19.19%
2013-01-25 05:17:59	144K	783212 kB	19.19%
2013-01-25 05:18:04	144K	783716 kB	19.19%
2013-01-25 05:18:09	144K	783616 kB	19.19%
2013-01-25 05:18:14	144K	784120 kB	19.19%
2013-01-25 05:18:19	144K	783616 kB	19.19%
2013-01-25 05:18:24	144K	783600 kB	19.19%
2013-01-25 05:18:29	144K	783692 kB	19.19%
2013-01-25 05:18:34	144K	783692 kB	19.19%
2013-01-25 05:18:39	144K	783940 kB	19.19%
2013-01-25 05:18:44	144K	784296 kB	19.19%
2013-01-25 05:18:49	144K	784428 kB	19.19%
2013-01-25 05:18:54	144K	784700 kB	19.19%
2013-01-25 05:18:59	144K	784832 kB	19.19%
2013-01-25 05:19:04	144K	784924 kB	19.19%
2013-01-25 05:19:09	144K	785056 kB	19.19%
2013-01-25 05:19:14	144K	785180 kB	19.19%
2013-01-25 05:19:19	144K	785560 kB	19.19%
2013-01-25 05:19:24	144K	785676 kB	19.19%
2013-01-25 05:19:29	144K	786304 kB	19.19%
2013-01-25 05:19:34	144K	785932 kB	19.19%
2013-01-25 05:19:39	144K	786180 kB	19.19%
2013-01-25 05:19:44	144K	786528 kB	19.19%
2013-01-25 05:19:49	144K	786676 kB	19.19%
2013-01-25 05:19:54	144K	786800 kB	19.19%
2013-01-25 05:19:59	144K	786676 kB	19.19%
2013-01-25 05:20:04	144K	787288 kB	19.19%
2013-01-25 05:20:09	144K	787040 kB	19.19%
2013-01-25 05:20:14	144K	787048 kB	19.19%
2013-01-25 05:20:19	144K	787048 kB	19.18%
2013-01-25 05:20:24	144K	787164 kB	19.18%
2013-01-25 05:20:29	144K	787180 kB	19.18%
2013-01-25 05:20:34	144K	788072 kB	19.18%
2013-01-25 05:20:39	144K	787824 kB	19.18%
2013-01-25 05:20:44	144K	787916 kB	19.18%
2013-01-25 05:20:49	144K	788040 kB	19.18%
2013-01-25 05:20:55	144K	788164 kB	19.18%
2013-01-25 05:21:00	144K	788412 kB	19.18%
2013-01-25 05:21:05	144K	788504 kB	19.18%
2013-01-25 05:21:10	144K	789016 kB	19.18%
2013-01-25 05:21:15	144K	788900 kB	19.18%
2013-01-25 05:21:20	144K	789024 kB	19.18%
2013-01-25 05:21:25	144K	789124 kB	19.18%
2013-01-25 05:21:30	144K	789140 kB	19.18%
2013-01-25 05:21:35	144K	789140 kB	19.18%
2013-01-25 05:21:40	144K	789396 kB	19.18%
2013-01-25 05:21:45	144K	789512 kB	19.18%
2013-01-25 05:21:50	144K	789644 kB	19.18%
2013-01-25 05:21:55	144K	789660 kB	19.18%
2013-01-25 05:22:00	144K	790156 kB	19.18%
2013-01-25 05:22:05	144K	790380 kB	19.18%
2013-01-25 05:22:10	144K	790388 kB	19.18%
2013-01-25 05:22:15	144K	790264 kB	19.18%
2013-01-25 05:22:20	144K	790636 kB	19.18%
2013-01-25 05:22:25	144K	790992 kB	19.18%
2013-01-25 05:22:30	144K	791140 kB	19.18%
2013-01-25 05:22:35	144K	791264 kB	19.18%
2013-01-25 05:22:40	144K	792008 kB	19.18%
2013-01-25 05:22:45	144K	792396 kB	19.18%
2013-01-25 05:22:50	144K	791900 kB	19.18%
2013-01-25 05:22:55	144K	792148 kB	19.18%
2013-01-25 05:23:00	144K	792124 kB	19.18%
2013-01-25 05:23:05	144K	792148 kB	19.18%
2013-01-25 05:23:10	144K	792644 kB	19.18%
2013-01-25 05:23:15	144K	792784 kB	19.18%
2013-01-25 05:23:20	144K	792992 kB	19.18%
2013-01-25 05:23:25	144K	792744 kB	19.18%
2013-01-25 05:23:30	144K	792752 kB	19.18%
2013-01-25 05:23:35	144K	794752 kB	19.18%
2013-01-25 05:23:40	144K	795000 kB	19.18%
2013-01-25 05:23:45	144K	795496 kB	19.18%
2013-01-25 05:23:50	144K	795496 kB	19.18%
2013-01-25 05:23:55	144K	795868 kB	19.18%
2013-01-25 05:24:00	144K	795852 kB	19.18%
2013-01-25 05:24:05	144K	796752 kB	19.18%
2013-01-25 05:24:10	144K	796488 kB	19.18%
2013-01-25 05:24:15	144K	796612 kB	19.18%
2013-01-25 05:24:20	144K	796456 kB	19.18%
2013-01-25 05:24:25	144K	796704 kB	19.18%
2013-01-25 05:24:30	144K	796588 kB	19.18%
2013-01-25 05:24:35	144K	796588 kB	19.18%
2013-01-25 05:24:40	144K	796960 kB	19.18%
2013-01-25 05:24:45	144K	797332 kB	19.18%
2013-01-25 05:24:51	144K	797340 kB	19.18%
2013-01-25 05:24:56	144K	797092 kB	19.18%
2013-01-25 05:25:01	144K	797580 kB	19.18%
2013-01-25 05:25:06	144K	798084 kB	19.18%
2013-01-25 05:25:11	144K	797852 kB	19.18%

TC3_H, Node A

TIMESTAMP	DISK USAGE	MEMORY USAGE	CPU USAGE
2013-01-25 10:31:48	317M	562792 kB	0.07%
2013-01-25 10:31:53	317M	577160 kB	0.07%
2013-01-25 10:31:58	317M	578228 kB	0.07%
2013-01-25 10:32:03	317M	578256 kB	0.07%
2013-01-25 10:32:08	317M	578332 kB	0.07%
2013-01-25 10:32:13	317M	578368 kB	0.07%
2013-01-25 10:32:18	317M	578144 kB	0.07%
2013-01-25 10:32:23	317M	578308 kB	0.07%
2013-01-25 10:32:28	317M	578116 kB	0.07%
2013-01-25 10:32:33	317M	578232 kB	0.07%
2013-01-25 10:32:38	317M	578344 kB	0.07%
2013-01-25 10:32:43	317M	578572 kB	0.07%
2013-01-25 10:32:48	317M	579276 kB	0.07%
2013-01-25 10:32:53	317M	579420 kB	0.07%
2013-01-25 10:32:58	317M	579300 kB	0.07%
2013-01-25 10:33:03	317M	579336 kB	0.07%
2013-01-25 10:33:08	317M	579136 kB	0.07%
2013-01-25 10:33:13	317M	579432 kB	0.07%
2013-01-25 10:33:18	317M	579336 kB	0.07%
2013-01-25 10:33:24	317M	579188 kB	0.07%
2013-01-25 10:33:29	317M	579100 kB	0.07%
2013-01-25 10:33:34	317M	579224 kB	0.07%
2013-01-25 10:33:39	317M	579248 kB	0.07%
2013-01-25 10:33:44	317M	579068 kB	0.07%
2013-01-25 10:33:49	317M	579248 kB	0.07%
2013-01-25 10:33:54	317M	579044 kB	0.07%
2013-01-25 10:33:59	317M	579172 kB	0.07%
2013-01-25 10:34:04	317M	579056 kB	0.07%
2013-01-25 10:34:09	317M	579380 kB	0.07%
2013-01-25 10:34:14	317M	579432 kB	0.07%
2013-01-25 10:34:19	317M	579372 kB	0.07%
2013-01-25 10:34:24	317M	579296 kB	0.07%
2013-01-25 10:34:29	317M	579236 kB	0.07%
2013-01-25 10:34:34	317M	579216 kB	0.07%
2013-01-25 10:34:39	317M	579328 kB	0.07%
2013-01-25 10:34:44	317M	579248 kB	0.07%
2013-01-25 10:34:49	317M	578992 kB	0.07%
2013-01-25 10:34:54	317M	579388 kB	0.07%
2013-01-25 10:34:59	317M	579360 kB	0.07%
2013-01-25 10:35:04	317M	579520 kB	0.07%
2013-01-25 10:35:09	317M	579636 kB	0.07%
2013-01-25 10:35:14	317M	579488 kB	0.07%
2013-01-25 10:35:20	317M	579108 kB	0.07%
2013-01-25 10:35:25	317M	579220 kB	0.07%
2013-01-25 10:35:30	317M	579332 kB	0.07%
2013-01-25 10:35:35	317M	579436 kB	0.07%
2013-01-25 10:35:40	317M	579404 kB	0.07%
2013-01-25 10:35:45	317M	579216 kB	0.07%
2013-01-25 10:35:50	317M	579588 kB	0.07%
2013-01-25 10:35:55	317M	579076 kB	0.07%
2013-01-25 10:36:00	317M	579464 kB	0.07%
2013-01-25 10:36:05	317M	579344 kB	0.07%
2013-01-25 10:36:10	317M	579480 kB	0.07%
2013-01-25 10:36:15	317M	579108 kB	0.07%
2013-01-25 10:36:20	317M	578868 kB	0.07%
2013-01-25 10:36:25	317M	579048 kB	0.07%
2013-01-25 10:36:30	317M	579020 kB	0.07%
2013-01-25 10:36:35	317M	579324 kB	0.07%
2013-01-25 10:36:40	317M	579224 kB	0.07%
2013-01-25 10:36:45	317M	579456 kB	0.07%
2013-01-25 10:36:50	317M	579300 kB	0.07%
2013-01-25 10:36:55	317M	579516 kB	0.07%
2013-01-25 10:37:00	317M	579532 kB	0.07%
2013-01-25 10:37:05	317M	579448 kB	0.07%
2013-01-25 10:37:10	317M	579424 kB	0.07%
2013-01-25 10:37:15	317M	579484 kB	0.07%
2013-01-25 10:37:20	317M	579544 kB	0.07%
2013-01-25 10:37:25	317M	579656 kB	0.07%
2013-01-25 10:37:31	317M	579700 kB	0.07%
2013-01-25 10:37:36	317M	579596 kB	0.07%
2013-01-25 10:37:41	317M	579352 kB	0.07%
2013-01-25 10:37:46	317M	579644 kB	0.07%
2013-01-25 10:37:51	317M	579508 kB	0.07%
2013-01-25 10:37:56	317M	579716 kB	0.07%
2013-01-25 10:38:01	317M	579584 kB	0.07%
2013-01-25 10:38:06	317M	579728 kB	0.07%
2013-01-25 10:38:11	317M	579552 kB	0.07%
2013-01-25 10:38:16	317M	579552 kB	0.07%
2013-01-25 10:38:21	317M	579960 kB	0.07%
2013-01-25 10:38:26	317M	579880 kB	0.07%
2013-01-25 10:38:31	317M	579744 kB	0.07%
2013-01-25 10:38:36	317M	579940 kB	0.07%
2013-01-25 10:38:41	317M	579672 kB	0.07%
2013-01-25 10:38:46	317M	579820 kB	0.07%
2013-01-25 10:38:51	317M	579756 kB	0.07%
2013-01-25 10:38:56	317M	579604 kB	0.07%

2013-01-25 10:39:01	317M	579864 kB	0.07%
2013-01-25 10:39:06	317M	580068 kB	0.07%
2013-01-25 10:39:11	317M	580268 kB	0.07%
2013-01-25 10:39:16	317M	580376 kB	0.07%
2013-01-25 10:39:21	317M	580056 kB	0.07%
2013-01-25 10:39:26	317M	580140 kB	0.07%
2013-01-25 10:39:31	317M	580200 kB	0.07%
2013-01-25 10:39:36	317M	580032 kB	0.07%
2013-01-25 10:39:42	317M	580040 kB	0.07%
2013-01-25 10:39:47	317M	580076 kB	0.07%
2013-01-25 10:39:52	317M	580128 kB	0.07%
2013-01-25 10:39:57	317M	580292 kB	0.07%
2013-01-25 10:40:02	317M	580192 kB	0.07%
2013-01-25 10:40:07	317M	580328 kB	0.07%
2013-01-25 10:40:12	317M	580328 kB	0.07%
2013-01-25 10:40:17	317M	580512 kB	0.07%
2013-01-25 10:40:22	317M	579880 kB	0.07%
2013-01-25 10:40:27	317M	580432 kB	0.07%
2013-01-25 10:40:32	317M	580300 kB	0.07%
2013-01-25 10:40:37	317M	580024 kB	0.07%
2013-01-25 10:40:42	317M	580128 kB	0.07%
2013-01-25 10:40:47	317M	580268 kB	0.07%
2013-01-25 10:40:52	317M	579712 kB	0.07%
2013-01-25 10:40:57	317M	580244 kB	0.07%
2013-01-25 10:41:02	317M	580032 kB	0.07%
2013-01-25 10:41:07	317M	580160 kB	0.07%
2013-01-25 10:41:12	317M	580352 kB	0.07%
2013-01-25 10:41:17	317M	580600 kB	0.07%
2013-01-25 10:41:22	317M	580500 kB	0.07%
2013-01-25 10:41:27	317M	580456 kB	0.07%
2013-01-25 10:41:32	317M	580632 kB	0.07%
2013-01-25 10:41:37	317M	580644 kB	0.07%
2013-01-25 10:41:42	317M	580820 kB	0.07%
2013-01-25 10:41:47	317M	580576 kB	0.07%
2013-01-25 10:41:53	317M	580392 kB	0.07%
2013-01-25 10:41:58	317M	580564 kB	0.07%
2013-01-25 10:42:03	317M	580076 kB	0.07%
2013-01-25 10:42:08	317M	580276 kB	0.07%
2013-01-25 10:42:13	317M	580384 kB	0.07%
2013-01-25 10:42:18	317M	580828 kB	0.07%
2013-01-25 10:42:23	317M	580700 kB	0.07%
2013-01-25 10:42:28	317M	580700 kB	0.07%
2013-01-25 10:42:33	317M	580460 kB	0.07%
2013-01-25 10:42:38	317M	580460 kB	0.07%
2013-01-25 10:42:43	317M	580408 kB	0.07%
2013-01-25 10:42:48	317M	580080 kB	0.07%
2013-01-25 10:42:53	317M	580548 kB	0.07%
2013-01-25 10:42:58	317M	580376 kB	0.07%
2013-01-25 10:43:03	317M	580408 kB	0.07%
2013-01-25 10:43:08	317M	580400 kB	0.07%
2013-01-25 10:43:13	317M	580460 kB	0.07%
2013-01-25 10:43:18	317M	580544 kB	0.07%
2013-01-25 10:43:23	317M	580460 kB	0.07%
2013-01-25 10:43:28	317M	580376 kB	0.07%
2013-01-25 10:43:33	317M	580488 kB	0.07%
2013-01-25 10:43:38	317M	580564 kB	0.07%
2013-01-25 10:43:43	317M	580552 kB	0.07%
2013-01-25 10:43:48	317M	580416 kB	0.07%
2013-01-25 10:43:53	317M	580388 kB	0.07%
2013-01-25 10:43:59	317M	580856 kB	0.07%
2013-01-25 10:44:04	317M	580664 kB	0.07%
2013-01-25 10:44:09	317M	580520 kB	0.07%
2013-01-25 10:44:14	317M	580848 kB	0.07%
2013-01-25 10:44:19	317M	580668 kB	0.07%
2013-01-25 10:44:24	317M	580640 kB	0.07%
2013-01-25 10:44:29	317M	580924 kB	0.07%
2013-01-25 10:44:34	317M	580716 kB	0.07%
2013-01-25 10:44:39	317M	580760 kB	0.07%
2013-01-25 10:44:44	317M	580944 kB	0.07%
2013-01-25 10:44:49	317M	580692 kB	0.07%
2013-01-25 10:44:54	317M	580500 kB	0.07%
2013-01-25 10:44:59	317M	580804 kB	0.07%
2013-01-25 10:45:04	317M	580752 kB	0.07%
2013-01-25 10:45:09	317M	580932 kB	0.07%
2013-01-25 10:45:14	317M	581036 kB	0.07%
2013-01-25 10:45:19	317M	580944 kB	0.07%
2013-01-25 10:45:24	317M	580932 kB	0.07%
2013-01-25 10:45:29	317M	580768 kB	0.07%
2013-01-25 10:45:34	317M	581028 kB	0.07%
2013-01-25 10:45:39	317M	580920 kB	0.07%
2013-01-25 10:45:44	317M	580852 kB	0.07%
2013-01-25 10:45:49	317M	581096 kB	0.07%
2013-01-25 10:45:54	317M	580992 kB	0.07%
2013-01-25 10:45:59	317M	580980 kB	0.07%
2013-01-25 10:46:04	317M	580804 kB	0.07%
2013-01-25 10:46:10	317M	581080 kB	0.07%
2013-01-25 10:46:15	317M	580876 kB	0.07%
2013-01-25 10:46:20	317M	581008 kB	0.07%
2013-01-25 10:46:25	317M	580996 kB	0.07%
2013-01-25 10:46:30	317M	580880 kB	0.07%
2013-01-25 10:46:35	317M	580876 kB	0.07%
2013-01-25 10:46:40	317M	580760 kB	0.07%
2013-01-25 10:46:45	317M	580624 kB	0.07%
2013-01-25 10:46:50	317M	580840 kB	0.07%
2013-01-25 10:46:55	317M	580932 kB	0.07%

TC3_H, Node B

TIMESTAMP	DISK USAGE	MEMORY USAGE	CPU USAGE
2013-01-25 05:57:49	197M	1184128 kB	18.11%
2013-01-25 05:57:54	197M	1197096 kB	18.11%
2013-01-25 05:57:59	197M	1197220 kB	18.11%
2013-01-25 05:58:04	197M	1198120 kB	18.11%
2013-01-25 05:58:09	197M	1197872 kB	18.11%
2013-01-25 05:58:14	197M	1197856 kB	18.11%
2013-01-25 05:58:19	197M	1197112 kB	18.11%
2013-01-25 05:58:24	197M	1197468 kB	18.11%
2013-01-25 05:58:29	197M	1197468 kB	18.11%
2013-01-25 05:58:34	197M	1197112 kB	18.11%
2013-01-25 05:58:39	197M	1197128 kB	18.11%
2013-01-25 05:58:44	197M	1197096 kB	18.11%
2013-01-25 05:58:49	197M	1197344 kB	18.11%
2013-01-25 05:58:55	197M	1197236 kB	18.11%
2013-01-25 05:59:00	197M	1197236 kB	18.11%
2013-01-25 05:59:05	197M	1196840 kB	18.11%
2013-01-25 05:59:10	197M	1196600 kB	18.11%
2013-01-25 05:59:15	197M	1197096 kB	18.11%
2013-01-25 05:59:20	197M	1196972 kB	18.11%
2013-01-25 05:59:25	197M	1197344 kB	18.11%
2013-01-25 05:59:30	197M	1197120 kB	18.11%
2013-01-25 05:59:35	197M	1197104 kB	18.11%
2013-01-25 05:59:40	197M	1197244 kB	18.11%
2013-01-25 05:59:45	197M	1196592 kB	18.11%
2013-01-25 05:59:50	197M	1196980 kB	18.11%
2013-01-25 05:59:55	197M	1196856 kB	18.11%
2013-01-25 06:00:00	197M	1197104 kB	18.11%
2013-01-25 06:00:05	197M	1196740 kB	18.11%
2013-01-25 06:00:10	197M	1197004 kB	18.11%
2013-01-25 06:00:15	197M	1196880 kB	18.11%
2013-01-25 06:00:20	197M	1196764 kB	18.11%
2013-01-25 06:00:25	197M	1196732 kB	18.11%
2013-01-25 06:00:30	197M	1196856 kB	18.11%
2013-01-25 06:00:35	197M	1196608 kB	18.11%
2013-01-25 06:00:40	197M	1197104 kB	18.11%
2013-01-25 06:00:45	197M	1196972 kB	18.11%
2013-01-25 06:00:50	197M	1196864 kB	18.11%
2013-01-25 06:00:55	197M	1196492 kB	18.11%
2013-01-25 06:01:00	197M	1197020 kB	18.11%
2013-01-25 06:01:05	197M	1197120 kB	18.11%
2013-01-25 06:01:10	197M	1197144 kB	18.11%
2013-01-25 06:01:15	197M	1197392 kB	18.11%
2013-01-25 06:01:20	197M	1197484 kB	18.11%
2013-01-25 06:01:25	197M	1197368 kB	18.11%
2013-01-25 06:01:30	197M	1197352 kB	18.11%
2013-01-25 06:01:35	197M	1197260 kB	18.11%
2013-01-25 06:01:40	197M	1197128 kB	18.11%
2013-01-25 06:01:45	197M	1199120 kB	18.11%
2013-01-25 06:01:50	197M	1198384 kB	18.11%
2013-01-25 06:01:55	197M	1198880 kB	18.11%
2013-01-25 06:02:00	197M	1198724 kB	18.11%
2013-01-25 06:02:05	197M	1198732 kB	18.1%
2013-01-25 06:02:10	197M	1198832 kB	18.1%
2013-01-25 06:02:15	197M	1198732 kB	18.1%
2013-01-25 06:02:20	197M	1198824 kB	18.1%
2013-01-25 06:02:25	197M	1198832 kB	18.1%
2013-01-25 06:02:30	197M	1198600 kB	18.1%
2013-01-25 06:02:35	197M	1198476 kB	18.1%
2013-01-25 06:02:40	197M	1198368 kB	18.1%
2013-01-25 06:02:45	197M	1198740 kB	18.1%
2013-01-25 06:02:50	197M	1198616 kB	18.1%
2013-01-25 06:02:56	197M	1198616 kB	18.1%
2013-01-25 06:03:01	197M	1198832 kB	18.1%
2013-01-25 06:03:06	197M	1198832 kB	18.1%
2013-01-25 06:03:11	197M	1198824 kB	18.1%
2013-01-25 06:03:16	197M	1199088 kB	18.1%
2013-01-25 06:03:21	197M	1198444 kB	18.1%
2013-01-25 06:03:26	197M	1199080 kB	18.1%
2013-01-25 06:03:31	197M	1200700 kB	18.1%
2013-01-25 06:03:36	197M	1199352 kB	18.1%
2013-01-25 06:03:41	197M	1199328 kB	18.1%
2013-01-25 06:03:46	197M	1199452 kB	18.1%
2013-01-25 06:03:51	197M	1199328 kB	18.1%
2013-01-25 06:03:56	197M	1199080 kB	18.1%
2013-01-25 06:04:01	197M	1199584 kB	18.1%
2013-01-25 06:04:06	197M	1199492 kB	18.1%
2013-01-25 06:04:11	197M	1199468 kB	18.1%
2013-01-25 06:04:16	197M	1199252 kB	18.1%
2013-01-25 06:04:21	197M	1199220 kB	18.1%
2013-01-25 06:04:26	197M	1199616 kB	18.1%
2013-01-25 06:04:31	197M	1199616 kB	18.1%
2013-01-25 06:04:36	197M	1199368 kB	18.1%
2013-01-25 06:04:41	197M	1199088 kB	18.1%
2013-01-25 06:04:46	197M	1199608 kB	18.1%
2013-01-25 06:04:51	197M	1199128 kB	18.1%
2013-01-25 06:04:56	197M	1199128 kB	18.1%

2013-01-25 06:05:01	197M	1199220 kB	18.1%
2013-01-25 06:05:06	197M	1199344 kB	18.1%
2013-01-25 06:05:11	197M	1199080 kB	18.1%
2013-01-25 06:05:16	197M	1199088 kB	18.1%
2013-01-25 06:05:21	197M	1199220 kB	18.1%
2013-01-25 06:05:26	197M	1198856 kB	18.1%
2013-01-25 06:05:31	197M	1199476 kB	18.1%
2013-01-25 06:05:36	197M	1199196 kB	18.1%
2013-01-25 06:05:41	197M	1198988 kB	18.1%
2013-01-25 06:05:46	197M	1199484 kB	18.1%
2013-01-25 06:05:51	197M	1199376 kB	18.1%
2013-01-25 06:05:56	197M	1199096 kB	18.1%
2013-01-25 06:06:01	197M	1199740 kB	18.1%
2013-01-25 06:06:06	197M	1199492 kB	18.1%
2013-01-25 06:06:11	197M	1199104 kB	18.1%
2013-01-25 06:06:16	197M	1198948 kB	18.1%
2013-01-25 06:06:21	197M	1199444 kB	18.1%
2013-01-25 06:06:26	197M	1199568 kB	18.1%
2013-01-25 06:06:31	197M	1199220 kB	18.1%
2013-01-25 06:06:36	197M	1199320 kB	18.1%
2013-01-25 06:06:41	197M	1199072 kB	18.1%
2013-01-25 06:06:46	197M	1199220 kB	18.1%
2013-01-25 06:06:52	197M	1199368 kB	18.1%
2013-01-25 06:06:57	197M	1198964 kB	18.1%
2013-01-25 06:07:02	197M	1199088 kB	18.1%
2013-01-25 06:07:07	197M	1199468 kB	18.1%
2013-01-25 06:07:12	197M	1199220 kB	18.1%
2013-01-25 06:07:17	197M	1199188 kB	18.1%
2013-01-25 06:07:22	197M	1199832 kB	18.1%
2013-01-25 06:07:27	197M	1199336 kB	18.1%
2013-01-25 06:07:32	197M	1199212 kB	18.1%
2013-01-25 06:07:37	197M	1199328 kB	18.1%
2013-01-25 06:07:42	197M	1199080 kB	18.1%
2013-01-25 06:07:47	197M	1199328 kB	18.1%
2013-01-25 06:07:52	197M	1199460 kB	18.1%
2013-01-25 06:07:57	197M	1199676 kB	18.1%
2013-01-25 06:08:02	197M	1199832 kB	18.1%
2013-01-25 06:08:07	197M	1199352 kB	18.1%
2013-01-25 06:08:12	197M	1199336 kB	18.1%
2013-01-25 06:08:17	197M	1199552 kB	18.1%
2013-01-25 06:08:22	197M	1199320 kB	18.1%
2013-01-25 06:08:27	197M	1199452 kB	18.1%
2013-01-25 06:08:32	197M	1199452 kB	18.1%
2013-01-25 06:08:37	197M	1199560 kB	18.09%
2013-01-25 06:08:42	197M	1199560 kB	18.09%
2013-01-25 06:08:47	197M	1199312 kB	18.09%
2013-01-25 06:08:52	197M	1199436 kB	18.09%
2013-01-25 06:08:57	197M	1199528 kB	18.09%
2013-01-25 06:09:02	197M	1199172 kB	18.09%
2013-01-25 06:09:07	197M	1199668 kB	18.09%
2013-01-25 06:09:12	197M	1199768 kB	18.09%
2013-01-25 06:09:17	197M	1199784 kB	18.09%
2013-01-25 06:09:22	197M	1199668 kB	18.09%
2013-01-25 06:09:27	197M	1199420 kB	18.09%
2013-01-25 06:09:32	197M	1199436 kB	18.09%
2013-01-25 06:09:37	197M	1199652 kB	18.09%
2013-01-25 06:09:42	197M	1199304 kB	18.09%
2013-01-25 06:09:47	197M	1199056 kB	18.09%
2013-01-25 06:09:52	197M	1199196 kB	18.09%
2013-01-25 06:09:57	197M	1199164 kB	18.09%
2013-01-25 06:10:02	197M	1200080 kB	18.09%
2013-01-25 06:10:07	197M	1199716 kB	18.09%
2013-01-25 06:10:12	197M	1199824 kB	18.09%
2013-01-25 06:10:17	197M	1199552 kB	18.09%
2013-01-25 06:10:22	197M	1199444 kB	18.09%
2013-01-25 06:10:27	197M	1199444 kB	18.09%
2013-01-25 06:10:32	197M	1199708 kB	18.09%
2013-01-25 06:10:37	197M	1199948 kB	18.09%
2013-01-25 06:10:42	197M	1200072 kB	18.09%
2013-01-25 06:10:47	197M	1199956 kB	18.09%
2013-01-25 06:10:53	197M	1199956 kB	18.09%
2013-01-25 06:10:58	197M	1199304 kB	18.09%
2013-01-25 06:11:03	197M	1199428 kB	18.09%
2013-01-25 06:11:08	197M	1199808 kB	18.09%
2013-01-25 06:11:13	197M	1199808 kB	18.09%
2013-01-25 06:11:18	197M	1199900 kB	18.09%
2013-01-25 06:11:23	197M	1199528 kB	18.09%
2013-01-25 06:11:28	197M	1199900 kB	18.09%
2013-01-25 06:11:33	197M	1200148 kB	18.09%
2013-01-25 06:11:38	197M	1199536 kB	18.09%
2013-01-25 06:11:43	197M	1200056 kB	18.09%
2013-01-25 06:11:48	197M	1200080 kB	18.09%
2013-01-25 06:11:53	197M	1200048 kB	18.09%
2013-01-25 06:11:58	197M	1199948 kB	18.09%
2013-01-25 06:12:03	197M	1200468 kB	18.09%
2013-01-25 06:12:08	197M	1200244 kB	18.09%
2013-01-25 06:12:13	197M	1200196 kB	18.09%
2013-01-25 06:12:18	197M	1199948 kB	18.09%
2013-01-25 06:12:23	197M	1200072 kB	18.09%
2013-01-25 06:12:28	197M	1199948 kB	18.09%
2013-01-25 06:12:33	197M	1199916 kB	18.09%
2013-01-25 06:12:38	197M	1200056 kB	18.09%
2013-01-25 06:12:43	197M	1200180 kB	18.09%
2013-01-25 06:12:48	197M	1200196 kB	18.09%
2013-01-25 06:12:53	197M	1200040 kB	18.09%

TC4, Node A

TIMESTAMP	DISK USAGE	MEMORY USAGE	CPU USAGE
2013-01-25 10:55:32	317M	777316 kB	0.07%
2013-01-25 10:55:37	317M	783772 kB	0.07%
2013-01-25 10:55:42	317M	843256 kB	0.07%
2013-01-25 10:55:47	317M	878208 kB	0.07%
2013-01-25 10:55:53	317M	877596 kB	0.07%
2013-01-25 10:55:58	317M	875916 kB	0.07%
2013-01-25 10:56:03	317M	879240 kB	0.07%
2013-01-25 10:56:09	317M	882092 kB	0.07%
2013-01-25 10:56:14	317M	885416 kB	0.07%
2013-01-25 10:56:19	317M	888516 kB	0.07%
2013-01-25 10:56:24	317M	891872 kB	0.07%
2013-01-25 10:56:29	317M	894492 kB	0.07%
2013-01-25 10:56:35	317M	897576 kB	0.07%
2013-01-25 10:56:40	317M	900460 kB	0.07%
2013-01-25 10:56:45	317M	903280 kB	0.07%
2013-01-25 10:56:50	317M	906016 kB	0.07%
2013-01-25 10:56:56	317M	909108 kB	0.07%
2013-01-25 10:57:01	317M	913556 kB	0.07%
2013-01-25 10:57:06	317M	916416 kB	0.07%
2013-01-25 10:57:11	317M	919300 kB	0.07%
2013-01-25 10:57:16	317M	922088 kB	0.07%
2013-01-25 10:57:22	317M	924096 kB	0.07%
2013-01-25 10:57:27	317M	926468 kB	0.07%
2013-01-25 10:57:32	317M	929452 kB	0.07%
2013-01-25 10:57:38	317M	932048 kB	0.07%
2013-01-25 10:57:43	317M	933900 kB	0.07%
2013-01-25 10:57:48	317M	936132 kB	0.07%
2013-01-25 10:57:53	317M	938340 kB	0.07%
2013-01-25 10:57:58	317M	941488 kB	0.07%
2013-01-25 10:58:03	317M	943704 kB	0.07%
2013-01-25 10:58:08	317M	946076 kB	0.07%
2013-01-25 10:58:14	317M	947920 kB	0.07%
2013-01-25 10:58:19	317M	949516 kB	0.07%
2013-01-25 10:58:24	317M	952144 kB	0.07%
2013-01-25 10:58:29	317M	954368 kB	0.07%
2013-01-25 10:58:34	317M	955964 kB	0.07%
2013-01-25 10:58:39	317M	957824 kB	0.07%
2013-01-25 10:58:45	317M	959436 kB	0.07%
2013-01-25 10:58:50	317M	961420 kB	0.07%
2013-01-25 10:58:55	317M	962908 kB	0.07%
2013-01-25 10:59:00	317M	964760 kB	0.07%
2013-01-25 10:59:05	317M	966636 kB	0.07%
2013-01-25 10:59:10	317M	968984 kB	0.07%
2013-01-25 10:59:15	317M	970836 kB	0.07%
2013-01-25 10:59:21	317M	973208 kB	0.07%
2013-01-25 10:59:26	317M	975068 kB	0.07%
2013-01-25 10:59:31	317M	977184 kB	0.07%
2013-01-25 10:59:36	317M	979128 kB	0.07%
2013-01-25 10:59:41	317M	981508 kB	0.07%
2013-01-25 10:59:46	317M	983624 kB	0.07%
2013-01-25 10:59:52	317M	986080 kB	0.07%
2013-01-25 10:59:57	317M	988072 kB	0.07%
2013-01-25 11:00:02	317M	990180 kB	0.07%
2013-01-25 11:00:07	317M	991552 kB	0.07%
2013-01-25 11:00:12	317M	993420 kB	0.07%
2013-01-25 11:00:18	317M	995148 kB	0.07%
2013-01-25 11:00:23	317M	935728 kB	0.07%
2013-01-25 11:00:28	317M	935860 kB	0.07%
2013-01-25 11:00:33	317M	936044 kB	0.07%
2013-01-25 11:00:38	317M	936408 kB	0.07%
2013-01-25 11:00:43	317M	936548 kB	0.07%
2013-01-25 11:00:48	317M	936896 kB	0.07%
2013-01-25 11:00:53	317M	937028 kB	0.07%
2013-01-25 11:00:58	317M	937004 kB	0.07%
2013-01-25 11:01:03	317M	937152 kB	0.07%
2013-01-25 11:01:08	317M	937136 kB	0.07%
2013-01-25 11:01:13	317M	937384 kB	0.07%
2013-01-25 11:01:18	317M	937748 kB	0.07%
2013-01-25 11:01:23	317M	937460 kB	0.07%
2013-01-25 11:01:28	317M	937684 kB	0.07%
2013-01-25 11:01:33	317M	937816 kB	0.07%
2013-01-25 11:01:38	317M	938180 kB	0.07%
2013-01-25 11:01:43	317M	938288 kB	0.07%
2013-01-25 11:01:48	317M	938396 kB	0.07%
2013-01-25 11:01:53	317M	938552 kB	0.07%
2013-01-25 11:01:58	317M	939180 kB	0.07%
2013-01-25 11:02:03	317M	939420 kB	0.07%
2013-01-25 11:02:08	317M	939428 kB	0.07%
2013-01-25 11:02:14	317M	939668 kB	0.07%
2013-01-25 11:02:19	317M	939924 kB	0.07%
2013-01-25 11:02:24	317M	940040 kB	0.07%
2013-01-25 11:02:29	317M	940024 kB	0.07%
2013-01-25 11:02:34	317M	940660 kB	0.07%
2013-01-25 11:02:39	317M	941048 kB	0.07%
2013-01-25 11:02:44	317M	940752 kB	0.07%
2013-01-25 11:02:49	317M	941272 kB	0.07%

2013-01-25 11:02:54	317M	941140 kB	0.07%
2013-01-25 11:02:59	317M	941692 kB	0.07%
2013-01-25 11:03:04	317M	941916 kB	0.07%
2013-01-25 11:03:09	317M	942156 kB	0.07%
2013-01-25 11:03:14	317M	942528 kB	0.07%
2013-01-25 11:03:19	317M	942652 kB	0.07%
2013-01-25 11:03:24	317M	942876 kB	0.07%
2013-01-25 11:03:29	317M	942892 kB	0.07%
2013-01-25 11:03:34	317M	943000 kB	0.07%
2013-01-25 11:03:39	317M	943636 kB	0.07%
2013-01-25 11:03:44	317M	943876 kB	0.07%
2013-01-25 11:03:49	317M	944776 kB	0.07%
2013-01-25 11:03:54	317M	945132 kB	0.07%
2013-01-25 11:03:59	317M	945248 kB	0.07%
2013-01-25 11:04:04	317M	945248 kB	0.07%
2013-01-25 11:04:09	317M	945628 kB	0.07%
2013-01-25 11:04:14	317M	945992 kB	0.07%
2013-01-25 11:04:19	317M	946240 kB	0.07%
2013-01-25 11:04:25	317M	946604 kB	0.07%
2013-01-25 11:04:30	317M	947488 kB	0.07%
2013-01-25 11:04:35	317M	947744 kB	0.07%
2013-01-25 11:04:40	317M	947868 kB	0.07%
2013-01-25 11:04:45	317M	948124 kB	0.07%
2013-01-25 11:04:50	317M	948248 kB	0.07%
2013-01-25 11:04:55	317M	948876 kB	0.07%
2013-01-25 11:05:00	317M	949108 kB	0.07%
2013-01-25 11:05:05	317M	949264 kB	0.07%
2013-01-25 11:05:10	317M	949240 kB	0.07%
2013-01-25 11:05:15	317M	949108 kB	0.07%
2013-01-25 11:05:20	317M	949976 kB	0.07%
2013-01-25 11:05:25	317M	950248 kB	0.07%
2013-01-25 11:05:30	317M	949820 kB	0.07%
2013-01-25 11:05:35	317M	950340 kB	0.07%
2013-01-25 11:05:40	317M	950216 kB	0.07%
2013-01-25 11:05:45	317M	950720 kB	0.07%
2013-01-25 11:05:50	317M	950472 kB	0.07%
2013-01-25 11:05:55	317M	950588 kB	0.07%
2013-01-25 11:06:00	317M	951224 kB	0.07%
2013-01-25 11:06:05	317M	951572 kB	0.07%
2013-01-25 11:06:10	317M	951696 kB	0.07%
2013-01-25 11:06:15	317M	952192 kB	0.07%
2013-01-25 11:06:20	317M	952152 kB	0.07%
2013-01-25 11:06:25	317M	952316 kB	0.07%
2013-01-25 11:06:31	317M	952656 kB	0.07%
2013-01-25 11:06:36	317M	952812 kB	0.07%
2013-01-25 11:06:41	317M	953052 kB	0.07%
2013-01-25 11:06:46	317M	953184 kB	0.07%
2013-01-25 11:06:51	317M	953556 kB	0.07%
2013-01-25 11:06:56	317M	953704 kB	0.07%
2013-01-25 11:07:01	317M	953936 kB	0.07%
2013-01-25 11:07:06	317M	954060 kB	0.07%
2013-01-25 11:07:11	317M	954424 kB	0.07%
2013-01-25 11:07:16	317M	954564 kB	0.07%
2013-01-25 11:07:21	317M	954920 kB	0.07%
2013-01-25 11:07:26	317M	954772 kB	0.07%
2013-01-25 11:07:31	317M	955556 kB	0.07%
2013-01-25 11:07:36	317M	955688 kB	0.07%
2013-01-25 11:07:41	317M	955780 kB	0.07%
2013-01-25 11:07:46	317M	955912 kB	0.07%
2013-01-25 11:07:51	317M	955920 kB	0.07%
2013-01-25 11:07:56	317M	956292 kB	0.07%
2013-01-25 11:08:01	317M	956384 kB	0.07%
2013-01-25 11:08:06	317M	957028 kB	0.07%
2013-01-25 11:08:11	317M	957144 kB	0.07%
2013-01-25 11:08:16	317M	957152 kB	0.07%
2013-01-25 11:08:21	317M	957532 kB	0.07%
2013-01-25 11:08:26	317M	957516 kB	0.07%
2013-01-25 11:08:31	317M	957772 kB	0.07%
2013-01-25 11:08:36	317M	957896 kB	0.07%
2013-01-25 11:08:41	317M	957888 kB	0.07%
2013-01-25 11:08:47	317M	958260 kB	0.07%
2013-01-25 11:08:52	317M	958656 kB	0.07%
2013-01-25 11:08:57	317M	958384 kB	0.07%
2013-01-25 11:09:02	317M	958516 kB	0.07%
2013-01-25 11:09:07	317M	959004 kB	0.07%
2013-01-25 11:09:12	317M	959012 kB	0.07%
2013-01-25 11:09:17	317M	959400 kB	0.07%
2013-01-25 11:09:22	317M	959244 kB	0.07%
2013-01-25 11:09:27	317M	959516 kB	0.07%
2013-01-25 11:09:32	317M	959632 kB	0.07%
2013-01-25 11:09:37	317M	959896 kB	0.07%
2013-01-25 11:09:42	317M	960260 kB	0.07%
2013-01-25 11:09:47	317M	960392 kB	0.07%
2013-01-25 11:09:52	317M	960508 kB	0.07%
2013-01-25 11:09:57	317M	960764 kB	0.07%
2013-01-25 11:10:02	317M	960740 kB	0.07%
2013-01-25 11:10:07	317M	961128 kB	0.07%
2013-01-25 11:10:12	317M	960988 kB	0.07%
2013-01-25 11:10:17	317M	961136 kB	0.07%
2013-01-25 11:10:22	317M	961368 kB	0.07%
2013-01-25 11:10:27	317M	961740 kB	0.07%
2013-01-25 11:10:32	317M	962004 kB	0.07%
2013-01-25 11:10:37	317M	962120 kB	0.07%
2013-01-25 11:10:42	317M	962252 kB	0.07%
2013-01-25 11:10:47	317M	962228 kB	0.07%

TC4, Node B

TIMESTAMP	DISK USAGE	MEMORY USAGE	CPU USAGE
2013-01-25 06:21:36	197M	1203816 kB	18.08%
2013-01-25 06:21:41	197M	1222100 kB	18.07%
2013-01-25 06:21:46	197M	1234416 kB	18.07%
2013-01-25 06:21:51	197M	1236664 kB	18.06%
2013-01-25 06:21:56	197M	1238176 kB	18.06%
2013-01-25 06:22:01	197M	1246940 kB	18.05%
2013-01-25 06:22:06	197M	1253452 kB	18.05%
2013-01-25 06:22:11	197M	1253760 kB	18.04%
2013-01-25 06:22:16	197M	1254272 kB	18.04%
2013-01-25 06:22:21	197M	1254452 kB	18.03%
2013-01-25 06:22:26	197M	1255420 kB	18.03%
2013-01-25 06:22:31	197M	1264936 kB	18.02%
2013-01-25 06:22:36	197M	1276592 kB	18.01%
2013-01-25 06:22:41	197M	1278320 kB	18.01%
2013-01-25 06:22:46	197M	1278840 kB	18%
2013-01-25 06:22:51	197M	1279236 kB	18%
2013-01-25 06:22:56	197M	1279692 kB	17.99%
2013-01-25 06:23:01	197M	1280228 kB	17.98%
2013-01-25 06:23:06	197M	1280684 kB	17.98%
2013-01-25 06:23:11	197M	1280692 kB	17.97%
2013-01-25 06:23:16	197M	1281080 kB	17.96%
2013-01-25 06:23:21	197M	1280948 kB	17.96%
2013-01-25 06:23:26	197M	1281368 kB	17.95%
2013-01-25 06:23:31	197M	1281784 kB	17.94%
2013-01-25 06:23:36	197M	1281684 kB	17.94%
2013-01-25 06:23:41	197M	1281808 kB	17.93%
2013-01-25 06:23:46	197M	1282296 kB	17.93%
2013-01-25 06:23:51	197M	1282412 kB	17.92%
2013-01-25 06:23:56	197M	1282180 kB	17.91%
2013-01-25 06:24:01	197M	1285484 kB	17.91%
2013-01-25 06:24:06	197M	1285288 kB	17.9%
2013-01-25 06:24:11	197M	1285716 kB	17.89%
2013-01-25 06:24:16	197M	1285552 kB	17.89%
2013-01-25 06:24:21	197M	1285280 kB	17.88%
2013-01-25 06:24:27	197M	1285280 kB	17.88%
2013-01-25 06:24:32	197M	1285800 kB	17.87%
2013-01-25 06:24:37	197M	1285924 kB	17.86%
2013-01-25 06:24:42	197M	1286032 kB	17.86%
2013-01-25 06:24:47	197M	1286312 kB	17.85%
2013-01-25 06:24:52	197M	1286404 kB	17.85%
2013-01-25 06:24:57	197M	1286676 kB	17.84%
2013-01-25 06:25:02	197M	1288700 kB	17.84%
2013-01-25 06:25:07	197M	1287420 kB	17.83%
2013-01-25 06:25:12	197M	1287360 kB	17.82%
2013-01-25 06:25:17	197M	1287892 kB	17.82%
2013-01-25 06:25:22	197M	1288568 kB	17.81%
2013-01-25 06:25:27	197M	1288064 kB	17.81%
2013-01-25 06:25:32	197M	1288196 kB	17.8%
2013-01-25 06:25:37	197M	1288172 kB	17.79%
2013-01-25 06:25:42	197M	1288172 kB	17.78%
2013-01-25 06:25:47	197M	1288512 kB	17.78%
2013-01-25 06:25:52	197M	1288420 kB	17.77%
2013-01-25 06:25:57	197M	1288492 kB	17.76%
2013-01-25 06:26:02	197M	1289048 kB	17.76%
2013-01-25 06:26:07	197M	1289188 kB	17.75%
2013-01-25 06:26:12	197M	1289792 kB	17.74%
2013-01-25 06:26:17	197M	1289552 kB	17.74%
2013-01-25 06:26:22	197M	1289304 kB	17.73%
2013-01-25 06:26:27	197M	1289156 kB	17.73%
2013-01-25 06:26:32	197M	1289256 kB	17.72%
2013-01-25 06:26:37	197M	1289304 kB	17.71%
2013-01-25 06:26:42	197M	1289660 kB	17.71%
2013-01-25 06:26:47	197M	1290032 kB	17.7%
2013-01-25 06:26:52	197M	1290412 kB	17.7%
2013-01-25 06:26:57	197M	1290404 kB	17.69%
2013-01-25 06:27:02	197M	1290140 kB	17.68%
2013-01-25 06:27:07	197M	1290064 kB	17.68%
2013-01-25 06:27:12	197M	1290008 kB	17.67%
2013-01-25 06:27:17	197M	1290396 kB	17.67%
2013-01-25 06:27:22	197M	1290372 kB	17.66%
2013-01-25 06:27:27	197M	1290404 kB	17.65%
2013-01-25 06:27:32	197M	1290156 kB	17.65%
2013-01-25 06:27:37	197M	1289876 kB	17.64%
2013-01-25 06:27:42	197M	1290652 kB	17.63%
2013-01-25 06:27:47	197M	1290584 kB	17.63%
2013-01-25 06:27:52	197M	1290964 kB	17.62%
2013-01-25 06:27:57	197M	1290892 kB	17.61%
2013-01-25 06:28:02	197M	1290972 kB	17.61%
2013-01-25 06:28:08	197M	1291024 kB	17.6%
2013-01-25 06:28:13	197M	1291256 kB	17.6%
2013-01-25 06:28:18	197M	1291560 kB	17.59%
2013-01-25 06:28:23	197M	1291188 kB	17.59%
2013-01-25 06:28:28	197M	1291356 kB	17.58%
2013-01-25 06:28:33	197M	1291760 kB	17.58%
2013-01-25 06:28:38	197M	1291380 kB	17.57%
2013-01-25 06:28:43	197M	1291504 kB	17.56%

2013-01-25 06:28:48	197M	1291868 kB	17.56%
2013-01-25 06:28:53	197M	1292256 kB	17.55%
2013-01-25 06:28:58	197M	1292000 kB	17.55%
2013-01-25 06:29:03	197M	1292024 kB	17.54%
2013-01-25 06:29:08	197M	1291984 kB	17.54%
2013-01-25 06:29:13	197M	1292024 kB	17.53%
2013-01-25 06:29:18	197M	1292628 kB	17.53%
2013-01-25 06:29:23	197M	1292612 kB	17.52%
2013-01-25 06:29:28	197M	1292612 kB	17.51%
2013-01-25 06:29:33	197M	1292984 kB	17.51%
2013-01-25 06:29:38	197M	1292876 kB	17.5%
2013-01-25 06:29:43	197M	1292848 kB	17.5%
2013-01-25 06:29:48	197M	1293860 kB	17.49%
2013-01-25 06:29:53	197M	1293868 kB	17.48%
2013-01-25 06:29:58	197M	1293488 kB	17.48%
2013-01-25 06:30:03	197M	1294660 kB	17.47%
2013-01-25 06:30:08	197M	1294008 kB	17.47%
2013-01-25 06:30:13	197M	1293768 kB	17.46%
2013-01-25 06:30:18	197M	1293396 kB	17.46%
2013-01-25 06:30:23	197M	1293644 kB	17.45%
2013-01-25 06:30:28	197M	1293884 kB	17.45%
2013-01-25 06:30:33	197M	1294056 kB	17.44%
2013-01-25 06:30:38	197M	1294768 kB	17.44%
2013-01-25 06:30:43	197M	1295504 kB	17.43%
2013-01-25 06:30:48	197M	1295744 kB	17.43%
2013-01-25 06:30:53	197M	1296100 kB	17.42%
2013-01-25 06:30:58	197M	1296108 kB	17.41%
2013-01-25 06:31:03	197M	1296332 kB	17.41%
2013-01-25 06:31:08	197M	1296704 kB	17.4%
2013-01-25 06:31:13	197M	1296472 kB	17.39%
2013-01-25 06:31:18	197M	1296092 kB	17.39%
2013-01-25 06:31:23	197M	1296596 kB	17.38%
2013-01-25 06:31:28	197M	1297000 kB	17.38%
2013-01-25 06:31:33	197M	1296696 kB	17.37%
2013-01-25 06:31:38	197M	1297188 kB	17.37%
2013-01-25 06:31:44	197M	1297140 kB	17.36%
2013-01-25 06:31:49	197M	1296596 kB	17.35%
2013-01-25 06:31:54	197M	1297008 kB	17.35%
2013-01-25 06:31:59	197M	1297472 kB	17.34%
2013-01-25 06:32:04	197M	1298216 kB	17.34%
2013-01-25 06:32:09	197M	1298084 kB	17.33%
2013-01-25 06:32:14	197M	1297944 kB	17.33%
2013-01-25 06:32:19	197M	1298040 kB	17.32%
2013-01-25 06:32:24	197M	1297580 kB	17.31%
2013-01-25 06:32:29	197M	1297728 kB	17.31%
2013-01-25 06:32:34	197M	1297704 kB	17.3%
2013-01-25 06:32:39	197M	1297960 kB	17.3%
2013-01-25 06:32:44	197M	1298232 kB	17.29%
2013-01-25 06:32:49	197M	1298216 kB	17.29%
2013-01-25 06:32:54	197M	1298264 kB	17.28%
2013-01-25 06:32:59	197M	1298580 kB	17.28%
2013-01-25 06:33:04	197M	1298976 kB	17.27%
2013-01-25 06:33:09	197M	1299084 kB	17.26%
2013-01-25 06:33:14	197M	1298960 kB	17.26%
2013-01-25 06:33:19	197M	1298944 kB	17.25%
2013-01-25 06:33:24	197M	1298984 kB	17.25%
2013-01-25 06:33:29	197M	1299216 kB	17.24%
2013-01-25 06:33:34	197M	1299216 kB	17.23%
2013-01-25 06:33:39	197M	1299620 kB	17.23%
2013-01-25 06:33:44	197M	1299620 kB	17.22%
2013-01-25 06:33:49	197M	1299736 kB	17.22%
2013-01-25 06:33:54	197M	1299496 kB	17.21%
2013-01-25 06:33:59	197M	1299440 kB	17.21%
2013-01-25 06:34:04	197M	1300488 kB	17.2%
2013-01-25 06:34:09	197M	1300092 kB	17.19%
2013-01-25 06:34:14	197M	1299944 kB	17.19%
2013-01-25 06:34:19	197M	1300068 kB	17.18%
2013-01-25 06:34:24	197M	1300100 kB	17.18%
2013-01-25 06:34:29	197M	1300024 kB	17.17%
2013-01-25 06:34:34	197M	1300192 kB	17.17%
2013-01-25 06:34:39	197M	1300356 kB	17.16%
2013-01-25 06:34:44	197M	1300124 kB	17.15%
2013-01-25 06:34:49	197M	1300440 kB	17.15%
2013-01-25 06:34:54	197M	1300572 kB	17.14%
2013-01-25 06:34:59	197M	1300704 kB	17.14%
2013-01-25 06:35:04	197M	1300844 kB	17.13%
2013-01-25 06:35:09	197M	1301572 kB	17.13%
2013-01-25 06:35:14	197M	1301356 kB	17.12%
2013-01-25 06:35:19	197M	1301596 kB	17.12%
2013-01-25 06:35:25	197M	1301440 kB	17.11%
2013-01-25 06:35:30	197M	1301356 kB	17.11%
2013-01-25 06:35:35	197M	1301316 kB	17.1%
2013-01-25 06:35:40	197M	1301696 kB	17.09%
2013-01-25 06:35:45	197M	1301648 kB	17.09%
2013-01-25 06:35:50	197M	1301424 kB	17.08%
2013-01-25 06:35:55	197M	1301564 kB	17.07%
2013-01-25 06:36:00	197M	1301184 kB	17.07%
2013-01-25 06:36:05	197M	1301788 kB	17.06%
2013-01-25 06:36:10	197M	1301664 kB	17.06%
2013-01-25 06:36:15	197M	1302060 kB	17.05%
2013-01-25 06:36:20	197M	1302184 kB	17.05%
2013-01-25 06:36:25	197M	1301920 kB	17.04%
2013-01-25 06:36:30	197M	1302044 kB	17.04%
2013-01-25 06:36:35	197M	1301920 kB	17.03%
2013-01-25 06:36:40	197M	1302176 kB	17.02%

TC5_H, Node A

TIMESTAMP	DISK USAGE	MEMORY USAGE	CPU USAGE
2013-01-25 11:28:43	317M	946164 kB	0.07%
2013-01-25 11:28:48	317M	956468 kB	0.07%
2013-01-25 11:28:53	317M	956312 kB	0.07%
2013-01-25 11:28:58	317M	977344 kB	0.07%
2013-01-25 11:29:03	317M	978724 kB	0.07%
2013-01-25 11:29:08	317M	980476 kB	0.07%
2013-01-25 11:29:13	317M	980584 kB	0.07%
2013-01-25 11:29:19	317M	981732 kB	0.07%
2013-01-25 11:29:24	317M	986940 kB	0.07%
2013-01-25 11:29:29	317M	993536 kB	0.07%
2013-01-25 11:29:34	317M	994404 kB	0.07%
2013-01-25 11:29:39	317M	995776 kB	0.07%
2013-01-25 11:29:45	317M	997016 kB	0.07%
2013-01-25 11:29:50	317M	998016 kB	0.07%
2013-01-25 11:29:55	319M	1003076 kB	0.07%
2013-01-25 11:30:00	333M	1010400 kB	0.07%
2013-01-25 11:30:06	343M	1010344 kB	0.07%
2013-01-25 11:30:11	353M	1008536 kB	0.07%
2013-01-25 11:30:16	362M	1010556 kB	0.07%
2013-01-25 11:30:22	324M	978648 kB	0.07%
2013-01-25 11:30:27	324M	979316 kB	0.07%
2013-01-25 11:30:32	324M	980436 kB	0.07%
2013-01-25 11:30:37	324M	981636 kB	0.07%
2013-01-25 11:30:42	324M	982416 kB	0.07%
2013-01-25 11:30:48	324M	983256 kB	0.07%
2013-01-25 11:30:53	324M	984128 kB	0.07%
2013-01-25 11:30:58	324M	985052 kB	0.07%
2013-01-25 11:31:03	324M	985376 kB	0.07%
2013-01-25 11:31:08	324M	986764 kB	0.07%
2013-01-25 11:31:13	324M	987352 kB	0.07%
2013-01-25 11:31:18	324M	988476 kB	0.07%
2013-01-25 11:31:24	324M	989576 kB	0.07%
2013-01-25 11:31:29	324M	990420 kB	0.07%
2013-01-25 11:31:34	324M	991032 kB	0.07%
2013-01-25 11:31:39	324M	992048 kB	0.07%
2013-01-25 11:31:44	324M	992908 kB	0.07%
2013-01-25 11:31:49	324M	994264 kB	0.07%
2013-01-25 11:31:54	324M	995264 kB	0.07%
2013-01-25 11:32:00	324M	995784 kB	0.07%
2013-01-25 11:32:05	324M	996744 kB	0.07%
2013-01-25 11:32:10	324M	997480 kB	0.07%
2013-01-25 11:32:15	324M	998224 kB	0.07%
2013-01-25 11:32:20	324M	998836 kB	0.07%
2013-01-25 11:32:25	324M	1000100 kB	0.07%
2013-01-25 11:32:30	324M	1000968 kB	0.07%
2013-01-25 11:32:35	324M	1002324 kB	0.07%
2013-01-25 11:32:40	324M	983800 kB	0.07%
2013-01-25 11:32:45	324M	983648 kB	0.07%
2013-01-25 11:32:51	324M	983668 kB	0.07%
2013-01-25 11:32:56	324M	983492 kB	0.07%
2013-01-25 11:33:01	324M	983596 kB	0.07%
2013-01-25 11:33:06	324M	983472 kB	0.07%
2013-01-25 11:33:11	324M	983332 kB	0.07%
2013-01-25 11:33:16	324M	983352 kB	0.07%
2013-01-25 11:33:21	324M	983456 kB	0.07%
2013-01-25 11:33:26	324M	983480 kB	0.07%
2013-01-25 11:33:31	324M	983512 kB	0.07%
2013-01-25 11:33:36	324M	983300 kB	0.07%
2013-01-25 11:33:41	324M	982504 kB	0.07%
2013-01-25 11:33:46	324M	982720 kB	0.07%
2013-01-25 11:33:51	324M	982596 kB	0.07%
2013-01-25 11:33:56	324M	982544 kB	0.07%
2013-01-25 11:34:01	324M	982348 kB	0.07%
2013-01-25 11:34:06	324M	982448 kB	0.07%
2013-01-25 11:34:11	324M	982644 kB	0.07%
2013-01-25 11:34:16	324M	982592 kB	0.07%
2013-01-25 11:34:21	324M	982444 kB	0.07%
2013-01-25 11:34:26	324M	982556 kB	0.07%
2013-01-25 11:34:31	324M	982568 kB	0.07%
2013-01-25 11:34:36	324M	982452 kB	0.07%
2013-01-25 11:34:41	324M	982124 kB	0.07%
2013-01-25 11:34:46	324M	982188 kB	0.07%
2013-01-25 11:34:51	324M	982124 kB	0.07%
2013-01-25 11:34:57	324M	982100 kB	0.07%
2013-01-25 11:35:02	324M	982524 kB	0.07%
2013-01-25 11:35:07	324M	982636 kB	0.07%
2013-01-25 11:35:12	324M	982420 kB	0.07%
2013-01-25 11:35:17	324M	982660 kB	0.07%
2013-01-25 11:35:22	324M	982896 kB	0.07%
2013-01-25 11:35:27	324M	982668 kB	0.07%
2013-01-25 11:35:32	324M	982752 kB	0.07%
2013-01-25 11:35:37	324M	982764 kB	0.07%
2013-01-25 11:35:42	324M	982448 kB	0.07%
2013-01-25 11:35:47	324M	982380 kB	0.07%
2013-01-25 11:35:52	324M	982164 kB	0.07%
2013-01-25 11:35:57	324M	982180 kB	0.07%

2013-01-25 11:36:02	324M	982260 kB	0.07%
2013-01-25 11:36:07	324M	982500 kB	0.07%
2013-01-25 11:36:12	324M	982356 kB	0.07%
2013-01-25 11:36:17	324M	982364 kB	0.07%
2013-01-25 11:36:22	324M	982372 kB	0.07%
2013-01-25 11:36:27	324M	982460 kB	0.07%
2013-01-25 11:36:32	324M	982620 kB	0.07%
2013-01-25 11:36:37	324M	982680 kB	0.07%
2013-01-25 11:36:42	324M	982784 kB	0.07%
2013-01-25 11:36:47	324M	983028 kB	0.07%
2013-01-25 11:36:52	324M	983244 kB	0.07%
2013-01-25 11:36:57	324M	983052 kB	0.07%
2013-01-25 11:37:02	324M	983020 kB	0.07%
2013-01-25 11:37:07	324M	982984 kB	0.07%
2013-01-25 11:37:13	324M	983368 kB	0.07%
2013-01-25 11:37:18	324M	983092 kB	0.07%
2013-01-25 11:37:23	324M	983360 kB	0.07%
2013-01-25 11:37:28	324M	983264 kB	0.07%
2013-01-25 11:37:33	324M	983124 kB	0.07%
2013-01-25 11:37:38	324M	983620 kB	0.07%
2013-01-25 11:37:43	324M	983160 kB	0.07%
2013-01-25 11:37:48	324M	983320 kB	0.07%
2013-01-25 11:37:53	324M	983516 kB	0.07%
2013-01-25 11:37:58	324M	983312 kB	0.07%
2013-01-25 11:38:03	324M	983244 kB	0.07%
2013-01-25 11:38:08	324M	983300 kB	0.07%
2013-01-25 11:38:13	324M	983300 kB	0.07%
2013-01-25 11:38:18	324M	983268 kB	0.07%
2013-01-25 11:38:23	324M	983200 kB	0.07%
2013-01-25 11:38:28	324M	983268 kB	0.07%
2013-01-25 11:38:33	324M	983164 kB	0.07%
2013-01-25 11:38:38	324M	983268 kB	0.07%
2013-01-25 11:38:43	324M	983224 kB	0.07%
2013-01-25 11:38:48	324M	982964 kB	0.07%
2013-01-25 11:38:53	324M	983348 kB	0.07%
2013-01-25 11:38:58	324M	983200 kB	0.07%
2013-01-25 11:39:03	324M	983084 kB	0.07%
2013-01-25 11:39:08	324M	983332 kB	0.07%
2013-01-25 11:39:13	324M	983304 kB	0.07%
2013-01-25 11:39:18	324M	983420 kB	0.07%
2013-01-25 11:39:24	324M	983284 kB	0.07%
2013-01-25 11:39:29	324M	983528 kB	0.07%
2013-01-25 11:39:34	324M	983656 kB	0.07%
2013-01-25 11:39:39	324M	985088 kB	0.07%
2013-01-25 11:39:44	324M	984212 kB	0.07%
2013-01-25 11:39:49	324M	984112 kB	0.07%
2013-01-25 11:39:54	324M	984132 kB	0.07%
2013-01-25 11:39:59	324M	984000 kB	0.07%
2013-01-25 11:40:04	324M	983924 kB	0.07%
2013-01-25 11:40:09	324M	984136 kB	0.07%
2013-01-25 11:40:14	324M	984052 kB	0.07%
2013-01-25 11:40:19	324M	984144 kB	0.07%
2013-01-25 11:40:24	324M	984136 kB	0.07%
2013-01-25 11:40:29	324M	984176 kB	0.07%
2013-01-25 11:40:34	324M	984224 kB	0.07%
2013-01-25 11:40:39	324M	984144 kB	0.07%
2013-01-25 11:40:44	324M	983984 kB	0.07%
2013-01-25 11:40:49	324M	984016 kB	0.07%
2013-01-25 11:40:54	324M	984016 kB	0.07%
2013-01-25 11:40:59	324M	984016 kB	0.07%
2013-01-25 11:41:04	324M	984060 kB	0.07%
2013-01-25 11:41:09	324M	984276 kB	0.07%
2013-01-25 11:41:14	324M	984076 kB	0.07%
2013-01-25 11:41:19	324M	984128 kB	0.07%
2013-01-25 11:41:24	324M	984248 kB	0.07%
2013-01-25 11:41:29	324M	984180 kB	0.07%
2013-01-25 11:41:34	324M	984372 kB	0.07%
2013-01-25 11:41:40	324M	984124 kB	0.07%
2013-01-25 11:41:45	324M	983952 kB	0.07%
2013-01-25 11:41:50	324M	984108 kB	0.07%
2013-01-25 11:41:55	324M	984428 kB	0.07%
2013-01-25 11:42:00	324M	984452 kB	0.07%
2013-01-25 11:42:05	324M	984048 kB	0.07%
2013-01-25 11:42:10	324M	984332 kB	0.07%
2013-01-25 11:42:15	324M	983908 kB	0.07%
2013-01-25 11:42:20	324M	984408 kB	0.07%
2013-01-25 11:42:25	324M	984220 kB	0.07%
2013-01-25 11:42:30	324M	984200 kB	0.07%
2013-01-25 11:42:35	324M	984224 kB	0.07%
2013-01-25 11:42:40	324M	984364 kB	0.07%
2013-01-25 11:42:45	324M	984356 kB	0.07%
2013-01-25 11:42:50	324M	984448 kB	0.07%
2013-01-25 11:42:55	324M	984188 kB	0.07%
2013-01-25 11:43:00	324M	984476 kB	0.07%
2013-01-25 11:43:05	324M	984156 kB	0.07%
2013-01-25 11:43:10	324M	984176 kB	0.07%
2013-01-25 11:43:15	324M	984352 kB	0.07%
2013-01-25 11:43:20	324M	984428 kB	0.07%
2013-01-25 11:43:25	324M	984040 kB	0.07%
2013-01-25 11:43:30	324M	984176 kB	0.07%
2013-01-25 11:43:35	324M	984144 kB	0.07%
2013-01-25 11:43:40	324M	983960 kB	0.07%
2013-01-25 11:43:45	324M	983708 kB	0.07%
2013-01-25 11:43:50	324M	983784 kB	0.07%
2013-01-25 11:43:56	324M	983912 kB	0.07%

TC5_H, Node B

TIMESTAMP	DISK USAGE	MEMORY USAGE	CPU USAGE
2013-01-25 06:54:46	197M	1224312 kB	16.68%
2013-01-25 06:54:51	197M	1236248 kB	16.68%
2013-01-25 06:54:56	197M	1236784 kB	16.68%
2013-01-25 06:55:01	197M	1255848 kB	16.68%
2013-01-25 06:55:06	197M	1261752 kB	16.67%
2013-01-25 06:55:11	197M	1265016 kB	16.66%
2013-01-25 06:55:16	197M	1266636 kB	16.66%
2013-01-25 06:55:22	197M	1278748 kB	16.65%
2013-01-25 06:55:27	197M	1284020 kB	16.64%
2013-01-25 06:55:32	197M	1284704 kB	16.64%
2013-01-25 06:55:37	197M	1285332 kB	16.63%
2013-01-25 06:55:42	197M	1285548 kB	16.62%
2013-01-25 06:55:47	197M	1297336 kB	16.62%
2013-01-25 06:55:52	197M	1308356 kB	16.61%
2013-01-25 06:55:57	197M	1309016 kB	16.6%
2013-01-25 06:56:02	197M	1312340 kB	16.59%
2013-01-25 06:56:07	197M	1310952 kB	16.59%
2013-01-25 06:56:12	197M	1311100 kB	16.59%
2013-01-25 06:56:17	197M	1311192 kB	16.58%
2013-01-25 06:56:22	197M	1311076 kB	16.58%
2013-01-25 06:56:27	197M	1311208 kB	16.57%
2013-01-25 06:56:32	197M	1246312 kB	16.57%
2013-01-25 06:56:37	4.0K	707020 kB	16.57%
2013-01-25 06:56:42	4.0K	706824 kB	16.57%
2013-01-25 06:56:47	4.0K	706464 kB	16.57%
2013-01-25 06:56:52	4.0K	706096 kB	16.57%
2013-01-25 06:56:57	4.0K	706048 kB	16.57%
2013-01-25 06:57:02	4.0K	706008 kB	16.57%
2013-01-25 06:57:07	4.0K	706008 kB	16.57%
2013-01-25 06:57:12	4.0K	706256 kB	16.57%
2013-01-25 06:57:17	4.0K	706256 kB	16.57%
2013-01-25 06:57:22	4.0K	706248 kB	16.57%
2013-01-25 06:57:27	4.0K	706016 kB	16.57%
2013-01-25 06:57:32	4.0K	706272 kB	16.57%
2013-01-25 06:57:37	4.0K	706272 kB	16.57%
2013-01-25 06:57:42	4.0K	706364 kB	16.57%
2013-01-25 06:57:47	4.0K	706240 kB	16.57%
2013-01-25 06:57:52	4.0K	706000 kB	16.57%
2013-01-25 06:57:57	4.0K	705876 kB	16.57%
2013-01-25 06:58:02	4.0K	706148 kB	16.57%
2013-01-25 06:58:07	4.0K	706280 kB	16.57%
2013-01-25 06:58:12	4.0K	706148 kB	16.57%
2013-01-25 06:58:17	4.0K	706148 kB	16.57%
2013-01-25 06:58:22	4.0K	706116 kB	16.57%
2013-01-25 06:58:27	4.0K	706140 kB	16.57%
2013-01-25 06:58:32	4.0K	706040 kB	16.57%
2013-01-25 06:58:37	4.0K	706412 kB	16.57%
2013-01-25 06:58:42	4.0K	705760 kB	16.57%
2013-01-25 06:58:47	4.0K	705760 kB	16.57%
2013-01-25 06:58:52	4.0K	705884 kB	16.57%
2013-01-25 06:58:57	4.0K	706124 kB	16.57%
2013-01-25 06:59:02	4.0K	705892 kB	16.57%
2013-01-25 06:59:07	4.0K	705892 kB	16.57%
2013-01-25 06:59:12	4.0K	706140 kB	16.57%
2013-01-25 06:59:17	4.0K	705860 kB	16.57%
2013-01-25 06:59:23	4.0K	706008 kB	16.57%
2013-01-25 06:59:28	4.0K	705908 kB	16.57%
2013-01-25 06:59:33	4.0K	705908 kB	16.57%
2013-01-25 06:59:38	4.0K	705884 kB	16.57%
2013-01-25 06:59:43	4.0K	705768 kB	16.57%
2013-01-25 06:59:48	4.0K	705900 kB	16.57%
2013-01-25 06:59:53	4.0K	705652 kB	16.57%
2013-01-25 06:59:58	4.0K	705620 kB	16.57%
2013-01-25 07:00:03	4.0K	706172 kB	16.57%
2013-01-25 07:00:08	4.0K	706196 kB	16.57%
2013-01-25 07:00:13	4.0K	706056 kB	16.57%
2013-01-25 07:00:18	4.0K	705776 kB	16.57%
2013-01-25 07:00:23	4.0K	705776 kB	16.57%
2013-01-25 07:00:28	4.0K	705644 kB	16.57%
2013-01-25 07:00:33	4.0K	705784 kB	16.57%
2013-01-25 07:00:38	4.0K	705380 kB	16.57%
2013-01-25 07:00:43	4.0K	710108 kB	16.57%
2013-01-25 07:00:48	4.0K	710108 kB	16.57%
2013-01-25 07:00:53	4.0K	710232 kB	16.57%
2013-01-25 07:00:58	4.0K	710092 kB	16.57%
2013-01-25 07:01:03	4.0K	709984 kB	16.57%
2013-01-25 07:01:08	4.0K	709984 kB	16.57%
2013-01-25 07:01:13	4.0K	709736 kB	16.57%
2013-01-25 07:01:18	4.0K	710332 kB	16.57%
2013-01-25 07:01:23	4.0K	710084 kB	16.57%
2013-01-25 07:01:28	4.0K	710232 kB	16.57%
2013-01-25 07:01:33	4.0K	710124 kB	16.57%
2013-01-25 07:01:38	4.0K	710348 kB	16.57%
2013-01-25 07:01:43	4.0K	710248 kB	16.57%
2013-01-25 07:01:48	4.0K	710364 kB	16.57%
2013-01-25 07:01:53	4.0K	710116 kB	16.57%

2013-01-25 07:01:58	4.0K	710084 kB	16.57%
2013-01-25 07:02:03	4.0K	710916 kB	16.57%
2013-01-25 07:02:08	4.0K	710388 kB	16.57%
2013-01-25 07:02:13	4.0K	709636 kB	16.57%
2013-01-25 07:02:18	4.0K	709480 kB	16.57%
2013-01-25 07:02:23	4.0K	709356 kB	16.57%
2013-01-25 07:02:28	4.0K	709232 kB	16.57%
2013-01-25 07:02:33	4.0K	709108 kB	16.57%
2013-01-25 07:02:38	4.0K	708952 kB	16.57%
2013-01-25 07:02:43	4.0K	708960 kB	16.57%
2013-01-25 07:02:48	4.0K	709216 kB	16.57%
2013-01-25 07:02:53	4.0K	709216 kB	16.57%
2013-01-25 07:02:58	4.0K	708944 kB	16.57%
2013-01-25 07:03:03	4.0K	708728 kB	16.57%
2013-01-25 07:03:08	4.0K	708868 kB	16.57%
2013-01-25 07:03:13	4.0K	708960 kB	16.57%
2013-01-25 07:03:18	4.0K	708960 kB	16.57%
2013-01-25 07:03:23	4.0K	709216 kB	16.57%
2013-01-25 07:03:28	4.0K	709836 kB	16.57%
2013-01-25 07:03:34	4.0K	709572 kB	16.57%
2013-01-25 07:03:39	4.0K	709364 kB	16.57%
2013-01-25 07:03:44	4.0K	709844 kB	16.57%
2013-01-25 07:03:49	4.0K	709348 kB	16.57%
2013-01-25 07:03:54	4.0K	709440 kB	16.57%
2013-01-25 07:03:59	4.0K	709464 kB	16.57%
2013-01-25 07:04:04	4.0K	709224 kB	16.57%
2013-01-25 07:04:09	4.0K	709216 kB	16.57%
2013-01-25 07:04:14	4.0K	709192 kB	16.57%
2013-01-25 07:04:19	4.0K	709084 kB	16.57%
2013-01-25 07:04:24	4.0K	708712 kB	16.57%
2013-01-25 07:04:29	4.0K	709100 kB	16.57%
2013-01-25 07:04:34	4.0K	709084 kB	16.57%
2013-01-25 07:04:39	4.0K	709208 kB	16.57%
2013-01-25 07:04:44	4.0K	709208 kB	16.57%
2013-01-25 07:04:49	4.0K	709100 kB	16.57%
2013-01-25 07:04:54	4.0K	709192 kB	16.57%
2013-01-25 07:04:59	4.0K	708984 kB	16.57%
2013-01-25 07:05:04	4.0K	709000 kB	16.57%
2013-01-25 07:05:09	4.0K	708876 kB	16.57%
2013-01-25 07:05:14	4.0K	709100 kB	16.57%
2013-01-25 07:05:19	4.0K	708868 kB	16.57%
2013-01-25 07:05:24	4.0K	708992 kB	16.57%
2013-01-25 07:05:29	4.0K	709116 kB	16.57%
2013-01-25 07:05:34	4.0K	708960 kB	16.57%
2013-01-25 07:05:39	4.0K	709208 kB	16.57%
2013-01-25 07:05:44	4.0K	709464 kB	16.57%
2013-01-25 07:05:49	4.0K	709604 kB	16.57%
2013-01-25 07:05:54	4.0K	709448 kB	16.57%
2013-01-25 07:05:59	4.0K	709092 kB	16.57%
2013-01-25 07:06:04	4.0K	709728 kB	16.57%
2013-01-25 07:06:09	4.0K	709720 kB	16.57%
2013-01-25 07:06:14	4.0K	709812 kB	16.57%
2013-01-25 07:06:19	4.0K	709580 kB	16.57%
2013-01-25 07:06:24	4.0K	709712 kB	16.57%
2013-01-25 07:06:29	4.0K	709712 kB	16.57%
2013-01-25 07:06:34	4.0K	709688 kB	16.57%
2013-01-25 07:06:39	4.0K	709580 kB	16.57%
2013-01-25 07:06:44	4.0K	709712 kB	16.57%
2013-01-25 07:06:49	4.0K	709712 kB	16.57%
2013-01-25 07:06:54	4.0K	709076 kB	16.57%
2013-01-25 07:06:59	4.0K	709076 kB	16.57%
2013-01-25 07:07:04	4.0K	708828 kB	16.57%
2013-01-25 07:07:09	4.0K	708680 kB	16.57%
2013-01-25 07:07:14	4.0K	708952 kB	16.57%
2013-01-25 07:07:19	4.0K	709076 kB	16.57%
2013-01-25 07:07:24	4.0K	709448 kB	16.57%
2013-01-25 07:07:29	4.0K	709168 kB	16.57%
2013-01-25 07:07:34	4.0K	709068 kB	16.57%
2013-01-25 07:07:40	4.0K	708960 kB	16.57%
2013-01-25 07:07:45	4.0K	708976 kB	16.57%
2013-01-25 07:07:50	4.0K	709192 kB	16.57%
2013-01-25 07:07:55	4.0K	708960 kB	16.57%
2013-01-25 07:08:00	4.0K	709084 kB	16.57%
2013-01-25 07:08:05	4.0K	709372 kB	16.57%
2013-01-25 07:08:10	4.0K	709224 kB	16.57%
2013-01-25 07:08:15	4.0K	709092 kB	16.57%
2013-01-25 07:08:20	4.0K	709092 kB	16.57%
2013-01-25 07:08:25	4.0K	709108 kB	16.57%
2013-01-25 07:08:30	4.0K	709200 kB	16.57%
2013-01-25 07:08:35	4.0K	709200 kB	16.57%
2013-01-25 07:08:40	4.0K	709092 kB	16.57%
2013-01-25 07:08:45	4.0K	709340 kB	16.57%
2013-01-25 07:08:50	4.0K	709308 kB	16.57%
2013-01-25 07:08:55	4.0K	709200 kB	16.57%
2013-01-25 07:09:00	4.0K	709208 kB	16.57%
2013-01-25 07:09:05	4.0K	709092 kB	16.57%
2013-01-25 07:09:10	4.0K	709308 kB	16.57%
2013-01-25 07:09:15	4.0K	709076 kB	16.57%
2013-01-25 07:09:20	4.0K	709464 kB	16.57%
2013-01-25 07:09:25	4.0K	709720 kB	16.57%
2013-01-25 07:09:30	4.0K	709828 kB	16.57%
2013-01-25 07:09:35	4.0K	709828 kB	16.57%
2013-01-25 07:09:40	4.0K	709828 kB	16.57%
2013-01-25 07:09:45	4.0K	709712 kB	16.57%
2013-01-25 07:09:50	4.0K	709704 kB	16.57%

TC6, Node A

TIMESTAMP	DISK USAGE	MEMORY USAGE	CPU USAGE
2013-01-25 11:55:44	329M	743344 kB	0.07%
2013-01-25 11:55:49	329M	771512 kB	0.07%
2013-01-25 11:55:54	329M	786656 kB	0.07%
2013-01-25 11:55:59	350M	818044 kB	0.07%
2013-01-25 11:56:04	330M	801948 kB	0.07%
2013-01-25 11:56:09	333M	808388 kB	0.07%
2013-01-25 11:56:14	333M	810404 kB	0.07%
2013-01-25 11:56:19	335M	824476 kB	0.07%
2013-01-25 11:56:24	335M	826352 kB	0.07%
2013-01-25 11:56:29	357M	876084 kB	0.07%
2013-01-25 11:56:34	333M	855832 kB	0.07%
2013-01-25 11:56:39	333M	860204 kB	0.07%
2013-01-25 11:56:44	335M	864404 kB	0.07%
2013-01-25 11:56:49	335M	868496 kB	0.07%
2013-01-25 11:56:54	365M	906036 kB	0.07%
2013-01-25 11:56:59	337M	881192 kB	0.07%
2013-01-25 11:57:04	337M	885548 kB	0.07%
2013-01-25 11:57:09	340M	893600 kB	0.07%
2013-01-25 11:57:14	340M	894824 kB	0.07%
2013-01-25 11:57:19	369M	897872 kB	0.07%
2013-01-25 11:57:25	342M	877724 kB	0.07%
2013-01-25 11:57:30	342M	879352 kB	0.07%
2013-01-25 11:57:35	344M	888900 kB	0.07%
2013-01-25 11:57:40	346M	903740 kB	0.07%
2013-01-25 11:57:45	374M	947560 kB	0.07%
2013-01-25 11:57:50	402M	983356 kB	0.07%
2013-01-25 11:57:55	346M	932268 kB	0.07%
2013-01-25 11:58:00	349M	935860 kB	0.07%
2013-01-25 11:58:05	352M	946052 kB	0.07%
2013-01-25 11:58:10	380M	985344 kB	0.07%
2013-01-25 11:58:15	409M	1011100 kB	0.07%
2013-01-25 11:58:20	351M	954332 kB	0.07%
2013-01-25 11:58:25	353M	963012 kB	0.07%
2013-01-25 11:58:30	356M	974800 kB	0.07%
2013-01-25 11:58:35	356M	976156 kB	0.07%
2013-01-25 11:58:40	368M	997468 kB	0.07%
2013-01-25 11:58:45	359M	999784 kB	0.07%
2013-01-25 11:58:51	361M	1004240 kB	0.07%
2013-01-25 11:58:56	364M	1010232 kB	0.07%
2013-01-25 11:59:01	364M	1008532 kB	0.07%
2013-01-25 11:59:06	382M	993632 kB	0.07%
2013-01-25 11:59:11	368M	982404 kB	0.07%
2013-01-25 11:59:16	368M	985620 kB	0.07%
2013-01-25 11:59:21	371M	997416 kB	0.07%
2013-01-25 11:59:26	371M	998896 kB	0.07%
2013-01-25 11:59:31	389M	998140 kB	0.07%
2013-01-25 11:59:36	372M	982752 kB	0.07%
2013-01-25 11:59:41	374M	990624 kB	0.07%
2013-01-25 11:59:46	374M	998320 kB	0.07%
2013-01-25 11:59:51	377M	1002784 kB	0.07%
2013-01-25 11:59:56	387M	1010196 kB	0.07%
2013-01-25 12:00:01	376M	978932 kB	0.07%
2013-01-25 12:00:06	379M	983324 kB	0.07%
2013-01-25 12:00:11	379M	986912 kB	0.07%
2013-01-25 12:00:16	381M	997468 kB	0.07%
2013-01-25 12:00:21	387M	1004388 kB	0.07%
2013-01-25 12:00:26	414M	1008876 kB	0.07%
2013-01-25 12:00:31	383M	978964 kB	0.07%
2013-01-25 12:00:37	383M	985532 kB	0.07%
2013-01-25 12:00:42	386M	989616 kB	0.07%
2013-01-25 12:00:47	398M	1006952 kB	0.07%
2013-01-25 12:00:52	427M	1008824 kB	0.07%
2013-01-25 12:00:57	387M	948772 kB	0.07%
2013-01-25 12:01:02	390M	952636 kB	0.07%
2013-01-25 12:01:07	390M	955364 kB	0.07%
2013-01-25 12:01:12	416M	993448 kB	0.07%
2013-01-25 12:01:17	392M	966900 kB	0.07%
2013-01-25 12:01:22	392M	968272 kB	0.07%
2013-01-25 12:01:27	395M	972364 kB	0.07%
2013-01-25 12:01:32	400M	979060 kB	0.07%
2013-01-25 12:01:37	426M	1007092 kB	0.07%
2013-01-25 12:01:42	397M	959780 kB	0.07%
2013-01-25 12:01:47	397M	961112 kB	0.07%
2013-01-25 12:01:53	399M	965220 kB	0.07%
2013-01-25 12:01:58	405M	972908 kB	0.07%
2013-01-25 12:02:03	432M	1002296 kB	0.07%
2013-01-25 12:02:08	401M	955828 kB	0.07%
2013-01-25 12:02:13	401M	957364 kB	0.07%
2013-01-25 12:02:18	404M	961312 kB	0.07%
2013-01-25 12:02:23	411M	970380 kB	0.07%
2013-01-25 12:02:28	439M	1000900 kB	0.07%
2013-01-25 12:02:33	406M	949816 kB	0.07%
2013-01-25 12:02:38	406M	951152 kB	0.07%
2013-01-25 12:02:43	408M	922780 kB	0.07%
2013-01-25 12:02:48	411M	926976 kB	0.07%
2013-01-25 12:02:53	411M	928736 kB	0.07%

2013-01-25 12:02:58	413M	933068 kB	0.07%
2013-01-25 12:03:03	416M	937004 kB	0.07%
2013-01-25 12:03:08	416M	938712 kB	0.07%
2013-01-25 12:03:13	418M	942772 kB	0.07%
2013-01-25 12:03:18	418M	944260 kB	0.07%
2013-01-25 12:03:23	421M	948832 kB	0.07%
2013-01-25 12:03:28	423M	952892 kB	0.07%
2013-01-25 12:03:33	423M	954148 kB	0.07%
2013-01-25 12:03:38	426M	958308 kB	0.07%
2013-01-25 12:03:43	426M	959672 kB	0.07%
2013-01-25 12:03:48	453M	989604 kB	0.07%
2013-01-25 12:03:54	431M	968996 kB	0.07%
2013-01-25 12:03:59	431M	970516 kB	0.07%
2013-01-25 12:04:04	433M	974236 kB	0.07%
2013-01-25 12:04:09	434M	975972 kB	0.07%
2013-01-25 12:04:14	458M	1003236 kB	0.07%
2013-01-25 12:04:19	438M	980260 kB	0.07%
2013-01-25 12:04:24	439M	982684 kB	0.07%
2013-01-25 12:04:29	441M	953464 kB	0.07%
2013-01-25 12:04:34	441M	954824 kB	0.07%
2013-01-25 12:04:39	463M	979104 kB	0.07%
2013-01-25 12:04:44	446M	963800 kB	0.07%
2013-01-25 12:04:49	446M	965076 kB	0.07%
2013-01-25 12:04:54	449M	969112 kB	0.07%
2013-01-25 12:04:59	449M	970624 kB	0.07%
2013-01-25 12:05:04	468M	991696 kB	0.07%
2013-01-25 12:05:09	494M	1010548 kB	0.07%
2013-01-25 12:05:15	454M	958332 kB	0.07%
2013-01-25 12:05:20	456M	962728 kB	0.07%
2013-01-25 12:05:25	456M	964400 kB	0.07%
2013-01-25 12:05:30	473M	983336 kB	0.07%
2013-01-25 12:05:35	503M	1010596 kB	0.07%
2013-01-25 12:05:40	461M	947836 kB	0.07%
2013-01-25 12:05:45	464M	951996 kB	0.07%
2013-01-25 12:05:50	464M	953412 kB	0.07%
2013-01-25 12:05:55	466M	957604 kB	0.07%
2013-01-25 12:06:00	474M	967488 kB	0.07%
2013-01-25 12:06:05	469M	963096 kB	0.07%
2013-01-25 12:06:10	471M	967744 kB	0.07%
2013-01-25 12:06:15	471M	969076 kB	0.07%
2013-01-25 12:06:20	474M	940604 kB	0.07%
2013-01-25 12:06:25	482M	950868 kB	0.07%
2013-01-25 12:06:30	476M	946512 kB	0.07%
2013-01-25 12:06:35	479M	950296 kB	0.07%
2013-01-25 12:06:40	481M	954748 kB	0.07%
2013-01-25 12:06:45	481M	956352 kB	0.07%
2013-01-25 12:06:50	498M	975092 kB	0.07%
2013-01-25 12:06:55	484M	961668 kB	0.07%
2013-01-25 12:07:00	486M	965768 kB	0.07%
2013-01-25 12:07:06	488M	969904 kB	0.07%
2013-01-25 12:07:11	488M	971592 kB	0.07%
2013-01-25 12:07:16	506M	990952 kB	0.07%
2013-01-25 12:07:21	491M	978324 kB	0.07%
2013-01-25 12:07:26	494M	981920 kB	0.07%
2013-01-25 12:07:31	496M	986368 kB	0.07%
2013-01-25 12:07:36	496M	991560 kB	0.07%
2013-01-25 12:07:41	508M	1004828 kB	0.07%
2013-01-25 12:07:46	538M	1011020 kB	0.07%
2013-01-25 12:07:51	520M	1008508 kB	0.07%
2013-01-25 12:07:56	544M	1009108 kB	0.07%
2013-01-25 12:08:01	570M	993176 kB	0.07%
2013-01-25 12:08:06	593M	1008600 kB	0.07%
2013-01-25 12:08:11	617M	1009556 kB	0.07%
2013-01-25 12:08:17	640M	1012144 kB	0.07%
2013-01-25 12:08:22	449M	898844 kB	0.07%
2013-01-25 12:08:27	449M	899132 kB	0.07%
2013-01-25 12:08:32	449M	899576 kB	0.07%
2013-01-25 12:08:37	451M	902436 kB	0.07%
2013-01-25 12:08:42	452M	874508 kB	0.07%
2013-01-25 12:08:47	452M	874900 kB	0.07%
2013-01-25 12:08:52	452M	875136 kB	0.07%
2013-01-25 12:08:57	452M	875688 kB	0.07%
2013-01-25 12:09:02	454M	878844 kB	0.07%
2013-01-25 12:09:07	454M	879720 kB	0.07%
2013-01-25 12:09:12	454M	880436 kB	0.07%
2013-01-25 12:09:17	454M	881044 kB	0.07%
2013-01-25 12:09:22	457M	884468 kB	0.07%
2013-01-25 12:09:27	457M	884808 kB	0.07%
2013-01-25 12:09:32	457M	885292 kB	0.07%
2013-01-25 12:09:37	457M	886004 kB	0.07%
2013-01-25 12:09:43	459M	889532 kB	0.07%
2013-01-25 12:09:48	459M	890184 kB	0.07%
2013-01-25 12:09:53	459M	890528 kB	0.07%
2013-01-25 12:09:58	459M	891328 kB	0.07%
2013-01-25 12:10:03	461M	894440 kB	0.07%
2013-01-25 12:10:08	461M	895188 kB	0.07%
2013-01-25 12:10:13	461M	895908 kB	0.07%
2013-01-25 12:10:18	461M	896244 kB	0.07%
2013-01-25 12:10:23	464M	899900 kB	0.07%
2013-01-25 12:10:28	464M	900532 kB	0.07%
2013-01-25 12:10:33	464M	901152 kB	0.07%
2013-01-25 12:10:38	465M	902984 kB	0.07%
2013-01-25 12:10:43	466M	905248 kB	0.07%
2013-01-25 12:10:48	466M	905780 kB	0.07%
2013-01-25 12:10:53	466M	906580 kB	0.07%

TC6, Node B

TIMESTAMP	DISK USAGE	MEMORY USAGE	CPU USAGE
2013-01-25 07:21:45	204M	1238036 kB	16.49%
2013-01-25 07:21:50	204M	1238060 kB	16.49%
2013-01-25 07:21:55	204M	1281440 kB	16.48%
2013-01-25 07:22:00	204M	1282144 kB	16.47%
2013-01-25 07:22:05	204M	1293716 kB	16.45%
2013-01-25 07:22:10	204M	1295680 kB	16.44%
2013-01-25 07:22:15	204M	1309232 kB	16.42%
2013-01-25 07:22:20	204M	1332116 kB	16.41%
2013-01-25 07:22:25	204M	1334688 kB	16.39%
2013-01-25 07:22:30	214M	1347188 kB	16.38%
2013-01-25 07:22:35	214M	1350080 kB	16.37%
2013-01-25 07:22:40	214M	1373156 kB	16.35%
2013-01-25 07:22:45	214M	1375820 kB	16.34%
2013-01-25 07:22:50	214M	1378376 kB	16.32%
2013-01-25 07:22:55	237M	1454492 kB	16.31%
2013-01-25 07:23:00	210M	1439468 kB	16.3%
2013-01-25 07:23:05	216M	1426884 kB	16.28%
2013-01-25 07:23:10	216M	1427680 kB	16.27%
2013-01-25 07:23:15	216M	1427764 kB	16.26%
2013-01-25 07:23:20	216M	1427464 kB	16.24%
2013-01-25 07:23:25	216M	1439072 kB	16.23%
2013-01-25 07:23:30	225M	1451072 kB	16.21%
2013-01-25 07:23:35	225M	1476048 kB	16.2%
2013-01-25 07:23:40	225M	1478624 kB	16.19%
2013-01-25 07:23:45	225M	1480952 kB	16.17%
2013-01-25 07:23:50	245M	1530792 kB	16.16%
2013-01-25 07:23:55	225M	1549232 kB	16.15%
2013-01-25 07:24:00	225M	1550892 kB	16.13%
2013-01-25 07:24:05	225M	1553000 kB	16.12%
2013-01-25 07:24:10	225M	1558832 kB	16.1%
2013-01-25 07:24:15	235M	1570552 kB	16.09%
2013-01-25 07:24:21	238M	1565320 kB	16.08%
2013-01-25 07:24:26	238M	1568240 kB	16.07%
2013-01-25 07:24:31	238M	1569768 kB	16.05%
2013-01-25 07:24:36	238M	1583600 kB	16.04%
2013-01-25 07:24:41	238M	1617488 kB	16.03%
2013-01-25 07:24:46	248M	1630096 kB	16.01%
2013-01-25 07:24:51	248M	1632304 kB	16%
2013-01-25 07:24:56	248M	1637056 kB	15.99%
2013-01-25 07:25:01	248M	1638880 kB	15.97%
2013-01-25 07:25:06	269M	1688512 kB	15.96%
2013-01-25 07:25:11	255M	1699212 kB	15.94%
2013-01-25 07:25:16	255M	1700712 kB	15.93%
2013-01-25 07:25:21	255M	1711444 kB	15.91%
2013-01-25 07:25:26	255M	1713400 kB	15.9%
2013-01-25 07:25:31	265M	1723984 kB	15.89%
2013-01-25 07:25:36	268M	1717660 kB	15.88%
2013-01-25 07:25:41	268M	1719976 kB	15.86%
2013-01-25 07:25:46	268M	1721856 kB	15.85%
2013-01-25 07:25:51	268M	1737484 kB	15.84%
2013-01-25 07:25:56	268M	1740540 kB	15.82%
2013-01-25 07:26:01	309M	1813620 kB	15.81%
2013-01-25 07:26:06	268M	1775900 kB	15.8%
2013-01-25 07:26:11	268M	1780596 kB	15.78%
2013-01-25 07:26:16	268M	1782620 kB	15.77%
2013-01-25 07:26:21	278M	1807144 kB	15.76%
2013-01-25 07:26:26	278M	1809816 kB	15.74%
2013-01-25 07:26:31	278M	1811412 kB	15.73%
2013-01-25 07:26:36	278M	1837820 kB	15.71%
2013-01-25 07:26:41	278M	1840236 kB	15.7%
2013-01-25 07:26:46	288M	1872936 kB	15.68%
2013-01-25 07:26:51	308M	1880728 kB	15.67%
2013-01-25 07:26:56	282M	1867740 kB	15.66%
2013-01-25 07:27:01	282M	1869140 kB	15.64%
2013-01-25 07:27:06	282M	1876540 kB	15.63%
2013-01-25 07:27:11	292M	1888632 kB	15.61%
2013-01-25 07:27:16	292M	1905136 kB	15.6%
2013-01-25 07:27:21	292M	1907244 kB	15.59%
2013-01-25 07:27:26	292M	1909188 kB	15.57%
2013-01-25 07:27:31	292M	1934912 kB	15.56%
2013-01-25 07:27:36	302M	1947716 kB	15.55%
2013-01-25 07:27:42	302M	1976504 kB	15.53%
2013-01-25 07:27:47	302M	2003388 kB	15.52%
2013-01-25 07:27:52	302M	2004860 kB	15.51%
2013-01-25 07:27:57	311M	2017568 kB	15.49%
2013-01-25 07:28:02	311M	2020388 kB	15.48%
2013-01-25 07:28:07	315M	1993340 kB	15.47%
2013-01-25 07:28:12	315M	1995628 kB	15.45%
2013-01-25 07:28:17	315M	1997388 kB	15.44%
2013-01-25 07:28:22	315M	1999232 kB	15.43%
2013-01-25 07:28:27	325M	2011108 kB	15.41%
2013-01-25 07:28:32	325M	2013520 kB	15.4%
2013-01-25 07:28:37	325M	2016124 kB	15.38%
2013-01-25 07:28:42	325M	2018188 kB	15.37%
2013-01-25 07:28:47	330M	2026332 kB	15.36%
2013-01-25 07:28:52	335M	2032800 kB	15.34%

2013-01-25 07:28:57	335M	2035256 kB	15.33%
2013-01-25 07:29:02	335M	2036952 kB	15.31%
2013-01-25 07:29:07	335M	2039712 kB	15.3%
2013-01-25 07:29:12	372M	2039752 kB	15.28%
2013-01-25 07:29:17	348M	1946272 kB	15.27%
2013-01-25 07:29:22	348M	1948060 kB	15.26%
2013-01-25 07:29:27	348M	1950584 kB	15.24%
2013-01-25 07:29:32	348M	1951996 kB	15.23%
2013-01-25 07:29:37	348M	1953980 kB	15.21%
2013-01-25 07:29:42	357M	1965580 kB	15.2%
2013-01-25 07:29:47	357M	1967244 kB	15.18%
2013-01-25 07:29:52	357M	1969312 kB	15.17%
2013-01-25 07:29:57	357M	1972056 kB	15.16%
2013-01-25 07:30:02	367M	1985560 kB	15.14%
2013-01-25 07:30:07	367M	1987612 kB	15.13%
2013-01-25 07:30:12	367M	1989156 kB	15.11%
2013-01-25 07:30:17	367M	1990996 kB	15.11%
2013-01-25 07:30:22	367M	1992884 kB	15.1%
2013-01-25 07:30:27	377M	2007872 kB	15.08%
2013-01-25 07:30:32	380M	1980748 kB	15.07%
2013-01-25 07:30:37	380M	1982268 kB	15.05%
2013-01-25 07:30:42	380M	1983600 kB	15.04%
2013-01-25 07:30:47	380M	1984888 kB	15.03%
2013-01-25 07:30:52	380M	1987084 kB	15.02%
2013-01-25 07:30:57	390M	1999500 kB	15.01%
2013-01-25 07:31:02	390M	2001200 kB	14.99%
2013-01-25 07:31:08	390M	2003332 kB	14.98%
2013-01-25 07:31:13	390M	2005412 kB	14.97%
2013-01-25 07:31:18	400M	2018604 kB	14.96%
2013-01-25 07:31:23	448M	2040368 kB	14.95%
2013-01-25 07:31:28	438M	2024696 kB	14.93%
2013-01-25 07:31:33	482M	2038928 kB	14.92%
2013-01-25 07:31:38	524M	2038248 kB	14.9%
2013-01-25 07:31:43	575M	2040264 kB	14.89%
2013-01-25 07:31:48	351M	1776100 kB	14.87%
2013-01-25 07:31:53	351M	1778776 kB	14.86%
2013-01-25 07:31:58	351M	1780504 kB	14.84%
2013-01-25 07:32:03	351M	1783328 kB	14.83%
2013-01-25 07:32:08	351M	1784980 kB	14.81%
2013-01-25 07:32:13	361M	1796528 kB	14.8%
2013-01-25 07:32:18	361M	1799184 kB	14.78%
2013-01-25 07:32:23	361M	1801692 kB	14.77%
2013-01-25 07:32:28	361M	1803392 kB	14.76%
2013-01-25 07:32:33	393M	1839436 kB	14.74%
2013-01-25 07:32:38	371M	1819104 kB	14.73%
2013-01-25 07:32:43	371M	1820904 kB	14.71%
2013-01-25 07:32:48	371M	1823124 kB	14.7%
2013-01-25 07:32:53	381M	1835236 kB	14.68%
2013-01-25 07:32:58	381M	1837536 kB	14.67%
2013-01-25 07:33:03	384M	1810724 kB	14.66%
2013-01-25 07:33:08	384M	1824448 kB	14.64%
2013-01-25 07:33:13	384M	1827400 kB	14.63%
2013-01-25 07:33:18	384M	1829348 kB	14.62%
2013-01-25 07:33:23	407M	1853384 kB	14.6%
2013-01-25 07:33:28	393M	1841764 kB	14.59%
2013-01-25 07:33:33	393M	1843912 kB	14.57%
2013-01-25 07:33:38	393M	1845736 kB	14.56%
2013-01-25 07:33:43	393M	1847156 kB	14.55%
2013-01-25 07:33:48	403M	1857440 kB	14.55%
2013-01-25 07:33:53	403M	1827148 kB	14.54%
2013-01-25 07:33:58	403M	1827764 kB	14.53%
2013-01-25 07:34:03	403M	1828524 kB	14.53%
2013-01-25 07:34:08	403M	1829316 kB	14.53%
2013-01-25 07:34:13	403M	1830144 kB	14.52%
2013-01-25 07:34:18	403M	1830764 kB	14.52%
2013-01-25 07:34:23	403M	1831416 kB	14.51%
2013-01-25 07:34:28	403M	1832316 kB	14.51%
2013-01-25 07:34:34	403M	1833588 kB	14.5%
2013-01-25 07:34:39	403M	1834520 kB	14.5%
2013-01-25 07:34:44	412M	1844044 kB	14.49%
2013-01-25 07:34:49	412M	1845208 kB	14.49%
2013-01-25 07:34:54	412M	1845764 kB	14.48%
2013-01-25 07:34:59	412M	1846716 kB	14.48%
2013-01-25 07:35:04	417M	1819008 kB	14.47%
2013-01-25 07:35:09	417M	1819832 kB	14.47%
2013-01-25 07:35:14	417M	1820552 kB	14.46%
2013-01-25 07:35:19	417M	1821512 kB	14.46%
2013-01-25 07:35:24	417M	1822588 kB	14.45%
2013-01-25 07:35:29	417M	1823364 kB	14.45%
2013-01-25 07:35:34	417M	1824232 kB	14.44%
2013-01-25 07:35:39	417M	1825380 kB	14.44%
2013-01-25 07:35:44	417M	1825848 kB	14.43%
2013-01-25 07:35:49	417M	1826932 kB	14.43%
2013-01-25 07:35:54	417M	1827552 kB	14.42%
2013-01-25 07:35:59	426M	1839032 kB	14.42%
2013-01-25 07:36:04	426M	1840220 kB	14.41%
2013-01-25 07:36:09	426M	1840988 kB	14.4%
2013-01-25 07:36:14	426M	1841708 kB	14.4%
2013-01-25 07:36:19	426M	1841800 kB	14.39%
2013-01-25 07:36:24	426M	1842848 kB	14.39%
2013-01-25 07:36:29	426M	1843996 kB	14.38%
2013-01-25 07:36:34	426M	1844740 kB	14.38%
2013-01-25 07:36:39	426M	1845700 kB	14.37%
2013-01-25 07:36:44	426M	1847112 kB	14.37%
2013-01-25 07:36:49	435M	1857516 kB	14.36%

TC7_H, Node A

TIMESTAMP	DISK USAGE	MEMORY USAGE	CPU USAGE
2013-01-25 12:19:19	484M	896744 kB	0.07%
2013-01-25 12:19:24	484M	924940 kB	0.07%
2013-01-25 12:19:29	484M	958528 kB	0.07%
2013-01-25 12:19:34	495M	975292 kB	0.07%
2013-01-25 12:19:39	524M	1006184 kB	0.07%
2013-01-25 12:19:44	487M	967260 kB	0.07%
2013-01-25 12:19:49	489M	971092 kB	0.07%
2013-01-25 12:19:54	490M	978308 kB	0.07%
2013-01-25 12:19:59	507M	1009828 kB	0.07%
2013-01-25 12:20:04	530M	1009396 kB	0.07%
2013-01-25 12:20:10	558M	991032 kB	0.07%
2013-01-25 12:20:15	570M	1010836 kB	0.07%
2013-01-25 12:20:20	527M	952164 kB	0.07%
2013-01-25 12:20:25	553M	988676 kB	0.07%
2013-01-25 12:20:31	592M	1010132 kB	0.07%
2013-01-25 12:20:36	640M	1010268 kB	0.07%
2013-01-25 12:20:42	640M	1005408 kB	0.07%
2013-01-25 12:20:47	688M	1006460 kB	0.07%
2013-01-25 12:20:52	663M	957020 kB	0.07%
2013-01-25 12:20:57	704M	994144 kB	0.07%
2013-01-25 12:21:03	756M	1009656 kB	0.07%
2013-01-25 12:21:08	731M	989820 kB	0.07%
2013-01-25 12:21:14	772M	1008036 kB	0.07%
2013-01-25 12:21:19	819M	987652 kB	0.07%
2013-01-25 12:21:25	862M	1008608 kB	0.07%
2013-01-25 12:21:30	887M	1009064 kB	0.07%
2013-01-25 12:21:35	886M	978624 kB	0.07%
2013-01-25 12:21:41	898M	1007264 kB	0.07%
2013-01-25 12:21:46	944M	992488 kB	0.07%
2013-01-25 12:21:52	956M	1008184 kB	0.07%
2013-01-25 12:21:57	1013M	1009800 kB	0.07%
2013-01-25 12:22:02	1.1G	1009252 kB	0.07%
2013-01-25 12:22:08	1.1G	1010856 kB	0.07%
2013-01-25 12:22:13	1.1G	953960 kB	0.07%
2013-01-25 12:22:18	1.1G	988616 kB	0.07%
2013-01-25 12:22:23	1.1G	995800 kB	0.07%
2013-01-25 12:22:29	1.2G	1010448 kB	0.07%
2013-01-25 12:22:34	1.2G	1008748 kB	0.07%
2013-01-25 12:22:39	1.2G	1009832 kB	0.07%
2013-01-25 12:22:45	1.3G	1009784 kB	0.07%
2013-01-25 12:22:50	1.3G	1010704 kB	0.07%
2013-01-25 12:22:56	1.4G	1011020 kB	0.07%
2013-01-25 12:23:01	1.3G	944136 kB	0.07%
2013-01-25 12:23:06	1.4G	1010104 kB	0.07%
2013-01-25 12:23:12	1.4G	1009144 kB	0.07%
2013-01-25 12:23:17	1.3G	973880 kB	0.07%
2013-01-25 12:23:22	1.4G	1009856 kB	0.07%
2013-01-25 12:23:27	1.4G	1011004 kB	0.07%
2013-01-25 12:23:33	1.4G	1000624 kB	0.07%
2013-01-25 12:23:38	1.4G	1008932 kB	0.07%
2013-01-25 12:23:43	1.5G	1010780 kB	0.07%
2013-01-25 12:23:49	1.5G	1000064 kB	0.07%
2013-01-25 12:23:54	1.5G	1011044 kB	0.07%
2013-01-25 12:23:59	1.5G	1010020 kB	0.07%
2013-01-25 12:24:04	1.6G	1011220 kB	0.07%
2013-01-25 12:24:09	1.5G	984856 kB	0.07%
2013-01-25 12:24:15	1.5G	926152 kB	0.07%
2013-01-25 12:24:20	1.5G	987080 kB	0.07%
2013-01-25 12:24:25	1.6G	1010184 kB	0.07%
2013-01-25 12:24:30	1.6G	1008644 kB	0.07%
2013-01-25 12:24:36	1.4G	950932 kB	0.07%
2013-01-25 12:24:41	1.4G	944040 kB	0.07%
2013-01-25 12:24:46	1.4G	1009680 kB	0.07%
2013-01-25 12:24:51	1.4G	980664 kB	0.07%
2013-01-25 12:24:56	1.4G	1010240 kB	0.07%
2013-01-25 12:25:01	1.5G	1009392 kB	0.07%
2013-01-25 12:25:06	1.5G	1008776 kB	0.07%
2013-01-25 12:25:11	1.5G	1009712 kB	0.07%
2013-01-25 12:25:17	1.6G	1010556 kB	0.07%
2013-01-25 12:25:22	1.6G	1011060 kB	0.07%
2013-01-25 12:25:27	1.7G	1010424 kB	0.07%
2013-01-25 12:25:32	1.5G	890976 kB	0.07%
2013-01-25 12:25:37	1.5G	953784 kB	0.07%
2013-01-25 12:25:42	1.5G	1009220 kB	0.07%
2013-01-25 12:25:47	1.6G	1009828 kB	0.07%
2013-01-25 12:25:52	1.6G	1010628 kB	0.07%
2013-01-25 12:25:57	1.6G	1010480 kB	0.07%
2013-01-25 12:26:02	1.6G	1010256 kB	0.07%
2013-01-25 12:26:07	911M	784860 kB	0.07%
2013-01-25 12:26:12	911M	784336 kB	0.07%
2013-01-25 12:26:17	911M	784500 kB	0.07%
2013-01-25 12:26:22	911M	784224 kB	0.07%
2013-01-25 12:26:27	911M	784188 kB	0.07%
2013-01-25 12:26:32	911M	784400 kB	0.07%
2013-01-25 12:26:37	911M	784400 kB	0.07%
2013-01-25 12:26:43	911M	784520 kB	0.07%

2013-01-25 12:26:48	911M	784256 kB	0.07%
2013-01-25 12:26:53	911M	784268 kB	0.07%
2013-01-25 12:26:58	911M	784244 kB	0.07%
2013-01-25 12:27:03	911M	784340 kB	0.07%
2013-01-25 12:27:08	911M	784408 kB	0.07%
2013-01-25 12:27:13	911M	784324 kB	0.07%
2013-01-25 12:27:18	911M	784188 kB	0.07%
2013-01-25 12:27:23	911M	784240 kB	0.07%
2013-01-25 12:27:28	911M	784284 kB	0.07%
2013-01-25 12:27:33	911M	784252 kB	0.07%
2013-01-25 12:27:38	911M	784384 kB	0.07%
2013-01-25 12:27:43	911M	784288 kB	0.07%
2013-01-25 12:27:48	911M	784488 kB	0.07%
2013-01-25 12:27:53	911M	784288 kB	0.07%
2013-01-25 12:27:58	911M	784296 kB	0.07%
2013-01-25 12:28:03	911M	784280 kB	0.07%
2013-01-25 12:28:08	911M	784320 kB	0.07%
2013-01-25 12:28:13	911M	784624 kB	0.07%
2013-01-25 12:28:18	911M	784376 kB	0.07%
2013-01-25 12:28:23	911M	784360 kB	0.07%
2013-01-25 12:28:28	911M	784368 kB	0.07%
2013-01-25 12:28:33	911M	784212 kB	0.07%
2013-01-25 12:28:38	911M	784816 kB	0.07%
2013-01-25 12:28:43	911M	784404 kB	0.07%
2013-01-25 12:28:48	911M	784500 kB	0.07%
2013-01-25 12:28:54	911M	784720 kB	0.07%
2013-01-25 12:28:59	911M	784572 kB	0.07%
2013-01-25 12:29:04	911M	784596 kB	0.07%
2013-01-25 12:29:09	911M	784740 kB	0.07%
2013-01-25 12:29:14	911M	784288 kB	0.07%
2013-01-25 12:29:19	911M	784564 kB	0.07%
2013-01-25 12:29:24	911M	785180 kB	0.07%
2013-01-25 12:29:29	911M	785304 kB	0.07%
2013-01-25 12:29:34	911M	785096 kB	0.07%
2013-01-25 12:29:39	911M	785260 kB	0.07%
2013-01-25 12:29:44	911M	785180 kB	0.07%
2013-01-25 12:29:49	911M	784968 kB	0.07%
2013-01-25 12:29:54	911M	784900 kB	0.07%
2013-01-25 12:29:59	911M	785236 kB	0.07%
2013-01-25 12:30:04	911M	785220 kB	0.07%
2013-01-25 12:30:09	911M	785236 kB	0.07%
2013-01-25 12:30:14	911M	785980 kB	0.07%
2013-01-25 12:30:19	911M	785908 kB	0.07%
2013-01-25 12:30:24	911M	785824 kB	0.07%
2013-01-25 12:30:29	911M	786128 kB	0.07%
2013-01-25 12:30:34	911M	786104 kB	0.07%
2013-01-25 12:30:39	911M	786220 kB	0.07%
2013-01-25 12:30:44	911M	786536 kB	0.07%
2013-01-25 12:30:49	911M	786396 kB	0.07%
2013-01-25 12:30:54	911M	786220 kB	0.07%
2013-01-25 12:30:59	911M	786372 kB	0.07%
2013-01-25 12:31:05	911M	786452 kB	0.07%
2013-01-25 12:31:10	911M	802120 kB	0.07%
2013-01-25 12:31:15	911M	802320 kB	0.07%
2013-01-25 12:31:20	911M	802344 kB	0.07%
2013-01-25 12:31:25	911M	802416 kB	0.07%
2013-01-25 12:31:30	911M	802404 kB	0.07%
2013-01-25 12:31:35	911M	802380 kB	0.07%
2013-01-25 12:31:40	911M	802232 kB	0.07%
2013-01-25 12:31:45	911M	802012 kB	0.07%
2013-01-25 12:31:50	911M	802352 kB	0.07%
2013-01-25 12:31:55	911M	802500 kB	0.07%
2013-01-25 12:32:00	911M	802636 kB	0.07%
2013-01-25 12:32:05	911M	802676 kB	0.07%
2013-01-25 12:32:10	911M	802944 kB	0.07%
2013-01-25 12:32:15	911M	802744 kB	0.07%
2013-01-25 12:32:20	911M	803000 kB	0.07%
2013-01-25 12:32:25	911M	802836 kB	0.07%
2013-01-25 12:32:30	911M	802876 kB	0.07%
2013-01-25 12:32:35	911M	802816 kB	0.07%
2013-01-25 12:32:40	911M	802900 kB	0.07%
2013-01-25 12:32:45	911M	802992 kB	0.07%
2013-01-25 12:32:50	911M	802780 kB	0.07%
2013-01-25 12:32:55	911M	802868 kB	0.07%
2013-01-25 12:33:00	911M	802800 kB	0.07%
2013-01-25 12:33:05	911M	802688 kB	0.07%
2013-01-25 12:33:10	911M	802884 kB	0.07%
2013-01-25 12:33:16	911M	802864 kB	0.07%
2013-01-25 12:33:21	911M	802840 kB	0.07%
2013-01-25 12:33:26	911M	803008 kB	0.07%
2013-01-25 12:33:31	911M	802976 kB	0.07%
2013-01-25 12:33:36	911M	803040 kB	0.07%
2013-01-25 12:33:41	911M	803068 kB	0.07%
2013-01-25 12:33:46	911M	803052 kB	0.07%
2013-01-25 12:33:51	911M	802948 kB	0.07%
2013-01-25 12:33:56	911M	803028 kB	0.07%
2013-01-25 12:34:01	911M	802888 kB	0.07%
2013-01-25 12:34:06	911M	802728 kB	0.07%
2013-01-25 12:34:11	911M	803988 kB	0.07%
2013-01-25 12:34:16	911M	804040 kB	0.07%
2013-01-25 12:34:21	911M	803604 kB	0.07%
2013-01-25 12:34:26	911M	803792 kB	0.07%
2013-01-25 12:34:31	911M	803836 kB	0.07%
2013-01-25 12:34:36	911M	803948 kB	0.07%
2013-01-25 12:34:41	911M	803592 kB	0.07%

TC7_H, Node B

TIMESTAMP	DISK USAGE	MEMORY USAGE	CPU USAGE
2013-01-25 07:45:23	459M	1876564 kB	14.2%
2013-01-25 07:45:28	459M	1888268 kB	14.2%
2013-01-25 07:45:33	459M	1926020 kB	14.2%
2013-01-25 07:45:38	459M	1940704 kB	14.18%
2013-01-25 07:45:43	459M	1943692 kB	14.17%
2013-01-25 07:45:48	459M	1946220 kB	14.15%
2013-01-25 07:45:53	468M	1958384 kB	14.14%
2013-01-25 07:45:58	468M	1965876 kB	14.12%
2013-01-25 07:46:03	472M	1975704 kB	14.11%
2013-01-25 07:46:08	475M	1952144 kB	14.09%
2013-01-25 07:46:13	475M	1954752 kB	14.08%
2013-01-25 07:46:19	504M	1988272 kB	14.06%
2013-01-25 07:46:24	481M	1970312 kB	14.05%
2013-01-25 07:46:29	491M	1984080 kB	14.03%
2013-01-25 07:46:34	491M	1996192 kB	14.01%
2013-01-25 07:46:39	491M	1999916 kB	14%
2013-01-25 07:46:44	513M	1993004 kB	13.98%
2013-01-25 07:46:49	530M	2015648 kB	13.96%
2013-01-25 07:46:54	530M	2020304 kB	13.95%
2013-01-25 07:46:59	539M	2034676 kB	13.93%
2013-01-25 07:47:04	539M	2037972 kB	13.91%
2013-01-25 07:47:09	549M	2038476 kB	13.9%
2013-01-25 07:47:14	549M	2040644 kB	13.88%
2013-01-25 07:47:19	562M	2038056 kB	13.86%
2013-01-25 07:47:24	610M	2023744 kB	13.85%
2013-01-25 07:47:29	554M	1960592 kB	13.84%
2013-01-25 07:47:34	522M	1930868 kB	13.82%
2013-01-25 07:47:39	532M	1944560 kB	13.81%
2013-01-25 07:47:44	532M	1948512 kB	13.79%
2013-01-25 07:47:49	532M	1951464 kB	13.78%
2013-01-25 07:47:54	542M	1984696 kB	13.76%
2013-01-25 07:47:59	542M	2009056 kB	13.75%
2013-01-25 07:48:04	564M	2037128 kB	13.73%
2013-01-25 07:48:09	555M	1985596 kB	13.72%
2013-01-25 07:48:14	555M	2003856 kB	13.71%
2013-01-25 07:48:19	555M	2022180 kB	13.7%
2013-01-25 07:48:24	565M	2041080 kB	13.68%
2013-01-25 07:48:29	565M	2038492 kB	13.66%
2013-01-25 07:48:34	589M	2039644 kB	13.65%
2013-01-25 07:48:39	579M	1987152 kB	13.63%
2013-01-25 07:48:44	625M	2036880 kB	13.61%
2013-01-25 07:48:49	679M	2013284 kB	13.6%
2013-01-25 07:48:54	722M	2039456 kB	13.58%
2013-01-25 07:48:59	767M	2038984 kB	13.56%
2013-01-25 07:49:04	535M	1801040 kB	13.56%
2013-01-25 07:49:09	535M	1800676 kB	13.56%
2013-01-25 07:49:15	535M	1800436 kB	13.55%
2013-01-25 07:49:20	535M	1800164 kB	13.55%
2013-01-25 07:49:25	535M	1799684 kB	13.55%
2013-01-25 07:49:30	4.0K	234596 kB	13.55%
2013-01-25 07:49:35	4.0K	233800 kB	13.55%
2013-01-25 07:49:40	4.0K	233364 kB	13.55%
2013-01-25 07:49:45	4.0K	233368 kB	13.55%
2013-01-25 07:49:50	4.0K	232776 kB	13.55%
2013-01-25 07:49:55	4.0K	232728 kB	13.55%
2013-01-25 07:50:00	4.0K	232844 kB	13.55%
2013-01-25 07:50:05	4.0K	233424 kB	13.55%
2013-01-25 07:50:10	4.0K	233260 kB	13.55%
2013-01-25 07:50:15	4.0K	233392 kB	13.55%
2013-01-25 07:50:20	4.0K	233144 kB	13.55%
2013-01-25 07:50:25	4.0K	232772 kB	13.55%
2013-01-25 07:50:30	4.0K	232740 kB	13.55%
2013-01-25 07:50:35	4.0K	232616 kB	13.55%
2013-01-25 07:50:40	4.0K	232616 kB	13.55%
2013-01-25 07:50:45	4.0K	232136 kB	13.55%
2013-01-25 07:50:50	4.0K	231888 kB	13.55%
2013-01-25 07:50:55	4.0K	231888 kB	13.55%
2013-01-25 07:51:00	4.0K	231888 kB	13.55%
2013-01-25 07:51:05	4.0K	231904 kB	13.55%
2013-01-25 07:51:10	4.0K	236600 kB	13.55%
2013-01-25 07:51:15	4.0K	236740 kB	13.55%
2013-01-25 07:51:20	4.0K	236740 kB	13.55%
2013-01-25 07:51:25	4.0K	236740 kB	13.55%
2013-01-25 07:51:30	4.0K	236584 kB	13.55%
2013-01-25 07:51:35	4.0K	236592 kB	13.55%
2013-01-25 07:51:40	4.0K	236608 kB	13.55%
2013-01-25 07:51:45	4.0K	236368 kB	13.55%
2013-01-25 07:51:50	4.0K	236212 kB	13.55%
2013-01-25 07:51:55	4.0K	236476 kB	13.55%
2013-01-25 07:52:00	4.0K	236608 kB	13.55%
2013-01-25 07:52:05	4.0K	237136 kB	13.55%
2013-01-25 07:52:10	4.0K	236740 kB	13.55%
2013-01-25 07:52:15	4.0K	236476 kB	13.55%
2013-01-25 07:52:20	4.0K	236236 kB	13.55%
2013-01-25 07:52:25	4.0K	236608 kB	13.55%
2013-01-25 07:52:30	4.0K	236576 kB	13.55%

2013-01-25 07:52:35	4.0K	236476 kB	13.55%
2013-01-25 07:52:40	4.0K	236352 kB	13.55%
2013-01-25 07:52:45	4.0K	236352 kB	13.55%
2013-01-25 07:52:50	4.0K	236328 kB	13.55%
2013-01-25 07:52:55	4.0K	236716 kB	13.55%
2013-01-25 07:53:00	4.0K	236476 kB	13.55%
2013-01-25 07:53:05	4.0K	236228 kB	13.55%
2013-01-25 07:53:10	4.0K	236212 kB	13.55%
2013-01-25 07:53:15	4.0K	236352 kB	13.55%
2013-01-25 07:53:21	4.0K	236640 kB	13.55%
2013-01-25 07:53:26	4.0K	236872 kB	13.55%
2013-01-25 07:53:31	4.0K	236716 kB	13.55%
2013-01-25 07:53:36	4.0K	236732 kB	13.55%
2013-01-25 07:53:41	4.0K	236616 kB	13.55%
2013-01-25 07:53:46	4.0K	236880 kB	13.55%
2013-01-25 07:53:51	4.0K	236856 kB	13.55%
2013-01-25 07:53:56	4.0K	237004 kB	13.55%
2013-01-25 07:54:01	4.0K	237012 kB	13.55%
2013-01-25 07:54:06	4.0K	237524 kB	13.55%
2013-01-25 07:54:11	4.0K	236796 kB	13.55%
2013-01-25 07:54:16	4.0K	237044 kB	13.55%
2013-01-25 07:54:21	4.0K	236920 kB	13.55%
2013-01-25 07:54:26	4.0K	236888 kB	13.55%
2013-01-25 07:54:31	4.0K	236516 kB	13.55%
2013-01-25 07:54:36	4.0K	237144 kB	13.55%
2013-01-25 07:54:41	4.0K	236780 kB	13.55%
2013-01-25 07:54:46	4.0K	237120 kB	13.55%
2013-01-25 07:54:51	4.0K	236888 kB	13.55%
2013-01-25 07:54:56	4.0K	236624 kB	13.55%
2013-01-25 07:55:01	4.0K	236664 kB	13.55%
2013-01-25 07:55:06	4.0K	236896 kB	13.55%
2013-01-25 07:55:11	4.0K	236664 kB	13.55%
2013-01-25 07:55:16	4.0K	237020 kB	13.55%
2013-01-25 07:55:21	4.0K	236648 kB	13.55%
2013-01-25 07:55:26	4.0K	236988 kB	13.55%
2013-01-25 07:55:31	4.0K	236740 kB	13.55%
2013-01-25 07:55:36	4.0K	236756 kB	13.55%
2013-01-25 07:55:41	4.0K	237004 kB	13.55%
2013-01-25 07:55:46	4.0K	236724 kB	13.55%
2013-01-25 07:55:51	4.0K	236740 kB	13.55%
2013-01-25 07:55:56	4.0K	236888 kB	13.55%
2013-01-25 07:56:01	4.0K	236912 kB	13.55%
2013-01-25 07:56:06	4.0K	237144 kB	13.55%
2013-01-25 07:56:11	4.0K	236772 kB	13.55%
2013-01-25 07:56:16	4.0K	237524 kB	13.55%
2013-01-25 07:56:21	4.0K	237400 kB	13.55%
2013-01-25 07:56:26	4.0K	237616 kB	13.55%
2013-01-25 07:56:31	4.0K	237476 kB	13.55%
2013-01-25 07:56:36	4.0K	237616 kB	13.55%
2013-01-25 07:56:41	4.0K	237144 kB	13.55%
2013-01-25 07:56:46	4.0K	237360 kB	13.55%
2013-01-25 07:56:51	4.0K	237740 kB	13.55%
2013-01-25 07:56:56	4.0K	237492 kB	13.55%
2013-01-25 07:57:01	4.0K	237244 kB	13.55%
2013-01-25 07:57:06	4.0K	237476 kB	13.55%
2013-01-25 07:57:11	4.0K	237996 kB	13.55%
2013-01-25 07:57:16	4.0K	237748 kB	13.55%
2013-01-25 07:57:21	4.0K	238120 kB	13.55%
2013-01-25 07:57:26	4.0K	237724 kB	13.55%
2013-01-25 07:57:31	4.0K	237980 kB	13.55%
2013-01-25 07:57:36	4.0K	237616 kB	13.55%
2013-01-25 07:57:41	4.0K	237624 kB	13.55%
2013-01-25 07:57:47	4.0K	236996 kB	13.55%
2013-01-25 07:57:52	4.0K	237252 kB	13.55%
2013-01-25 07:57:57	4.0K	237624 kB	13.55%
2013-01-25 07:58:02	4.0K	238408 kB	13.55%
2013-01-25 07:58:07	4.0K	238020 kB	13.55%
2013-01-25 07:58:12	4.0K	238416 kB	13.55%
2013-01-25 07:58:17	4.0K	238268 kB	13.55%
2013-01-25 07:58:22	4.0K	238268 kB	13.55%
2013-01-25 07:58:27	4.0K	237988 kB	13.54%
2013-01-25 07:58:32	4.0K	238112 kB	13.54%
2013-01-25 07:58:37	4.0K	238120 kB	13.54%
2013-01-25 07:58:42	4.0K	238120 kB	13.54%
2013-01-25 07:58:47	4.0K	238236 kB	13.54%
2013-01-25 07:58:52	4.0K	238608 kB	13.54%
2013-01-25 07:58:57	4.0K	238484 kB	13.54%
2013-01-25 07:59:02	4.0K	238748 kB	13.54%
2013-01-25 07:59:07	4.0K	238980 kB	13.54%
2013-01-25 07:59:12	4.0K	238732 kB	13.54%
2013-01-25 07:59:17	4.0K	238368 kB	13.54%
2013-01-25 07:59:22	4.0K	238244 kB	13.54%
2013-01-25 07:59:27	4.0K	238212 kB	13.54%
2013-01-25 07:59:32	4.0K	238212 kB	13.54%
2013-01-25 07:59:37	4.0K	238228 kB	13.54%
2013-01-25 07:59:42	4.0K	238700 kB	13.54%
2013-01-25 07:59:47	4.0K	238616 kB	13.54%
2013-01-25 07:59:52	4.0K	238500 kB	13.54%
2013-01-25 07:59:57	4.0K	238252 kB	13.54%
2013-01-25 08:00:02	4.0K	239384 kB	13.54%
2013-01-25 08:00:07	4.0K	238780 kB	13.54%
2013-01-25 08:00:12	4.0K	238144 kB	13.54%
2013-01-25 08:00:17	4.0K	238780 kB	13.54%
2013-01-25 08:00:22	4.0K	238252 kB	13.54%
2013-01-25 08:00:27	4.0K	238640 kB	13.54%