Cluster infrastructure for biological and health related research.

Sophia Corsava
Vladimir Getov
Harrow School of Computer Science

Cluster Infrastructure for Biological and Health Related Research

Sophia Corsava and Vladimir Getov
Harrow School of Computer Science, University of Westminster, London, U.K.
Email: sophiac6@yahoo.com, V.S.Getov@westminster.ac.uk

Abstract

Researchers in the biological and health industries need powerful and stable systems for their work. These systems must be dependable, fault-tolerant, highly available and easy to use. To cope with these demands we propose the use of computational and data clusters in a fail-over configuration combined with the grid technology and job scheduling. Our infrastructure has been deployed successfully for running time-critical applications in commercial environments. We also present experimental results from this pilot implementation that demonstrate the viability of our approach.

KEYWORDS: CLUSTER, BIOLOGICAL RESEARCH, HIGH AVAILABILITY, DISTRIBUTED, PARALLEL.

1. Introduction

Researchers, analysts, scientists and engineers, need reliable and powerful systems. Having the ability to run multiple analyses, experiments and realistic simulations can lead to new and more comprehensive discoveries. Research scientists in any industry can properly investigate new topics by performing experiments in a trial and error mode. However, researchers are often led to wrong conclusions as they are faced with processing problems. To overcome them, they reduce sample populations. Alternative scenarios cannot be investigated thoroughly and creativity cannot be fully expressed. Delivery of processing outputs and timing get adversely affected as well. Opportunities get lost this way too, if it takes months to determine which is the best molecular model for drug design for example. Data mining techniques cannot be put to their full use, as by nature they are processing intensive. The majority of database servers cannot withstand the load of running repeated comparisons of large data groups against a set of possible parameters and outcomes. Processing needs to be smooth, transparent and efficient.

Structured troubleshooting and fault correction approaches are widely used in the application domain. These techniques include recursive restarts [13], check pointing [17], reboot [13] and undoing old configurations [13]. The check-pointing technique allows applications to recover from the last point of failure by copying on a regular basis their status on stable storage and then retrieving it. Application recursive restarts are based on the principle of infrastructure-centric software design: move intelligence from endpoints into the supporting infrastructure. Reboot, restarts not only the application but also the underlying operating system and undoing old configurations involves restoring old backups and overwriting current assumed "invalid" settings. A newer approach is the N-layered architecture for application development that allows for resiliency and better performance [5].

A lot of important work has been done, in the areas of fault diagnosis, performance and decision-making. Current performance/diagnostic methods include: the threshold analysis, the bottleneck analysis, the what's different analysis and the correlation analysis. In this area, significant results have been reported by J. Hellerstein [10, 11]. Closely related to our project is also the very important work done by John Wilkes and R. Golding on self-managing, self-configuring storage [9, 16]. In addition, fault and/or decision trees are commonly used to diagnose problems and action corrective measures.

This paper is organized as follows. In section 2 we discuss current problems. In section 3 we present a proposed building methodology for biological and health related researches while section 4 presents results from one of our commercial implementations.

2. Current Problems

Parallel and distributed applications consist of multiple components. The sound inter-operability of these components is important for the correct functionality of the service as a whole. A common problem in large systems is that component interdependencies are highly complex and therefore difficult to maintain. These components may be front-end web services or application

GUIs, that users use to connect, and back-end processes such as databases and other applications for transaction processing. Very often, a component failure can have catastrophic results. For example, if the front-end web services fail, users cannot connect to the site and therefore cannot access the back-end database. In some cases there may be legal and ethical repercussions from distributed component failures [2, 3, 13]. Current problems for parallel and distributed applications could be summarized in four main categories as follows: 1) Users unable to access one or more service components. 2) Users can access service components, but they do not function properly (logical errors, bugs etc). 3) Users can access services but performance is bad. 4) Work in progress gets interrupted unexpectedly, because one or more service components (server, application, network) crash mid-way.

Measuring performance and system availability is a rather complicated task as collecting raw system utilisation statistics is hardly adequate to determine if the system has a performance problem or not and how available it is. Currently problems include, determining the optimum performance baseline per application/server, determining the diagnostic method and decision process to use, standardizing performance data collection, deciding which measurements to use, where to keep them, for how long and how to present them in a meaningful manner so they can be assessed. Performance problems can be classified in 4 main categories. 1) Progressive performance degradation, 2) Recurrent performance problems, 3) Expected Load spikes and 4) Unexpected performance problems. To effectively troubleshoot any problem, it is necessary to have correct input real data. This seems to be surprisingly difficult as most uptime claims are false [13], benchmarks are not realistic (as most do not consider random faults) [13], support people do not fully understand their systems and human experts are expensive and difficult to find [2, 3, 13]. Finally, as technology advances rapidly, more than one human experts need to work together to determine the problem.

3. Building Methodology

3.1 Infrastructure Design

To cope with the large amounts of computations, processing and data mining requirements in biological, and health researches, we propose the following hardware and software infrastructure design:

- The use of Unix-based systems (PCs, workstations or servers).
- The classification of applications in two major categories, computational (front end) and data processing (back end).
- The use of a high availability cluster software such as Veritas Cluster Server (VCS) [15].
- The clustering of servers together in major cluster groups using VCS, per operating system and application type. Machines belonging to the same cluster group should be built in exactly the same manner, so any one can be used if a machine in the cluster becomes unavailable. Floating IP addresses (Virtual IP addresses) per machine, type of application and cluster group can be used for additional fault-tolerance. Cluster groups can be addressed by DNS naming conventions to make them more easily remembered.
- The use of network based disk storage for all data and databases, so that data disks are commonly accessible by all servers that belong to the data processing group. These disks should also be clustered together in a high-availability fail-over configuration.
- The use of an additional network for backups and administration purposes.
- The use of load balancing scheduling facilities such as the Load Sharing Facility (LSF) [18], to be used for all types of scheduling data processing, data mining and batch jobs.
- The use of grid technology to make available resources amongst biological and health communities globally [7, 12, 18].
- The use of comprehensive performance measurement tools and reporting mechanisms.

Unix-based hosts have been proven more stable and more powerful than any other operating system type. For example a PC running linux is much more powerful compared to a PC running windows of the same hardware specification.

Application segregation per type makes administration and support much easier. In addition, security issues can be handled more efficiently, by isolating homogeneous hosts together, within a specified IP range or even an entirely separate network segment. Load balancing can be configured much easier amongst the members of an application cluster. In one of our commercial implementations for a database cluster, we had utilisation thresholds set to: CPU util=95%, Memory util=85%, Disk I/O Util=90% and Network Util=95%. When these were exceeded, an additional node automatically joined the database cluster.

Commercial high-availability cluster software such as Veritas Cluster server consists of the cluster agent software and kernel modules that integrate it to the operating system. Each cluster agent has 4 main components; the monitor that looks after the cluster, the cluster startup script, the cluster stop script and the cluster clean script that stops cleanly the cluster software. For each group there is one cluster master. Any cluster member can be made cluster master, if the active one fails. This is the default behaviour of the cluster software. It does not allow any configuration changes while it is running either on the application dynamically or on its own static and dynamic files. It determines the clustered application/host/resource/component status via the change of the heartbeat transmission over a dedicated private cluster network. The cluster heartbeat is determined by probing that takes place every Z seconds over a pair of dedicated network interface cards that cannot be used for anything else. When the heartbeat is disrupted the cluster software classifies the condition as a critical fault and fails-over services (resources/applications/components) to the next best available node. A fail-over is an action by which the next best host in a pre-configured list takes over the role of the failed one. In this way if a host or an application fails, the next host in the list will be nominated to start/restart clustered services with minimal service interruption. Such a failover can take from 1 to 5 minutes on average each time. Cluster groups can have up to 256 members. Clusters are inherently fault-tolerant by design. More information about the Veritas Cluster software Suite can be found in [15].

Let us consider a system that has only one component whose reliability is 60%. This means that the overall system reliability is also is 60%. If we have a cluster with N number of members working in parallel, the formula that would give us the cluster reliability would be [14]:

$$\begin{aligned} R_s &= 1 - Q_s = 1 - (Q_1 \cdot Q_2 \cdot \ldots \cdot Q_n) \\ &= 1 - [(1 - R_1) \cdot (1 - R_2) \cdot \ldots \cdot (1 - R_n)] \\ &= 1 - \prod_{i=1}^{n} (1 - R_i) \end{aligned}$$

$$Q_s = P(X_1)P(X_2)\ldots P(X_n)$$

Or

$$Q_s = \prod_{i=1}^{n} Q_i$$

Where

Q_s = Unreliability of the system,

X_i = Event of failure of unit *i*,

$P(X_i)$ = probability of failure of unit *i*.

Rs = cluster system reliability

In a cluster system as such, one node needs to succeed in order for the cluster to be considered successful. Table 1 shows how reliability increases with the addition of nodes based on the above formula.

Number of cluster nodes	Reliability
1	60%
2	84%
4	97%

Table 1 (System Reliability)- System reliability as a function of the number of components/cluster members.

We can clearly see that the reliability of a cluster system increases as more nodes as added to it, even if these nodes are not overly reliable themselves [14].

The use of network based disk storage ensures that if hosts become unavailable, data will still be available as they reside on different physical locations. Good examples of this type of disks are Network Appliances [19]. Disk devices as such are representative of NAS architectures. NAS devices enhance scalability by eliminating shared controllers and enable direct host access to potentially thousands of shared devices [9]. All data reside on shared disks and are accessible over the network via the NFS protocol. In each cluster group, any disk device can be used by any cluster member (see Figure 1). The SAN [16] technology is an even promising one, but it is not stable for production at the moment and it requires a lot of support and configuration.

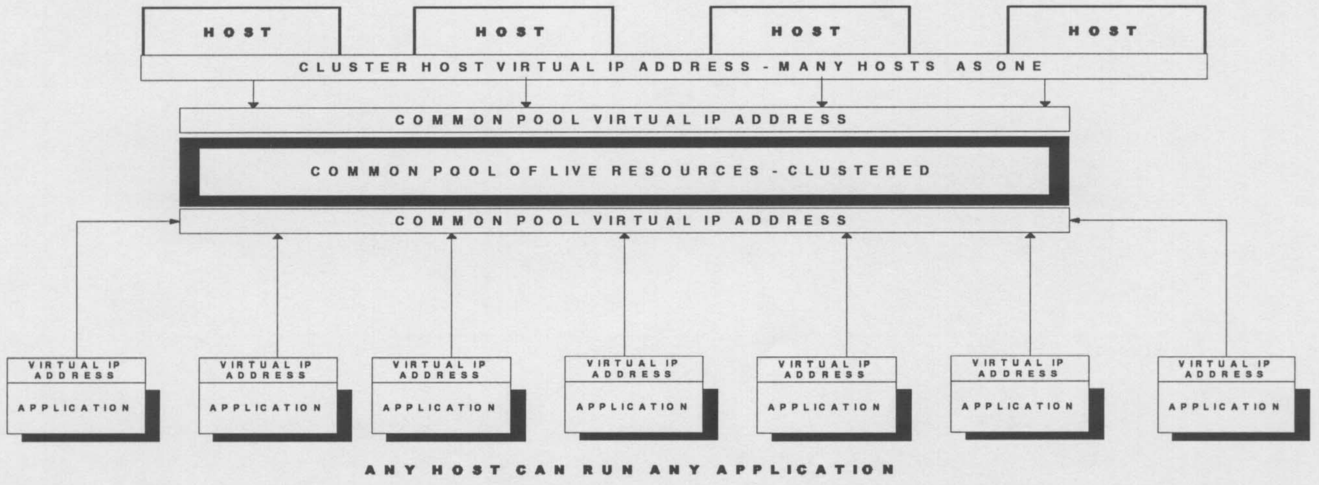
HOST
HOST
HOST
HOST
CLUSTER HOST VIRTUAL IP ADDRESS - MANY HOSTS AS ONE
COMMON POOL VIRTUAL IP ADDRESS
COMMON POOL OF LIVE RESOURCES - CLUSTERED
COMMON POOL VIRTUAL IP ADDRESS
VIRTUAL IP ADDRESS
APPLICATION
ANY HOST CAN RUN ANY APPLICATION

Figure 1. High Level view of a cluster with floating IP addresses (virtual IP addresses). Any host can use any resource.

Using an additional network for administration and backup/restore purposes lessens any performance/load overheads onto production network(s).

The latest version of the LSF [18] application by Platform is based on the grid technology. It is compatible with most operating systems such as Solaris, HP-UX, Linux, Windows, MAC-OS, etc. It allows the creation of virtual organisations with support for more than 100 clusters, 200,000 CPUs and 500,000 active jobs [18]. It comes with an easy to use user-friendly interface that allows users to schedule and monitor processing-intensive jobs. The grid engine, it is based on, enables users to choose from a variety of available hosts and resources, not only within a local site, but amongst different geographical sites. A toolkit such as Globus [20] can be used to present grid-enabled services and resources to users from different geographical sites. Combined with a high-availability cluster software such as Veritas Cluster Server, hosts are almost always available and some of the common problems (hosts disappearing in the middle of a computation, load balancing, application failures etc) do not affect computations and processing. Load balancing can be controlled from within LSF and the cluster software. They both allow the definition of load thresholds that are server and application specific. If these thresholds are exceeded, operators are notified and services are automatically moved over to the next best node or a new node joins the cluster [3, 4, 15, 18].

Network based common disks, allow for heterogeneous clusters to co-exist. In one of our commercial implementations, we had Linux and Windows clusters accessing the same data disks. To enable communication between the hosts, we used the SAMBA [6], utility so linux machines could "talk" to windows machines and vice versa. We developed software that could failover services amongst heterogeneous clusters, as the restriction of most commercial cluster software is that only hosts of the same operating system type can be clustered together.

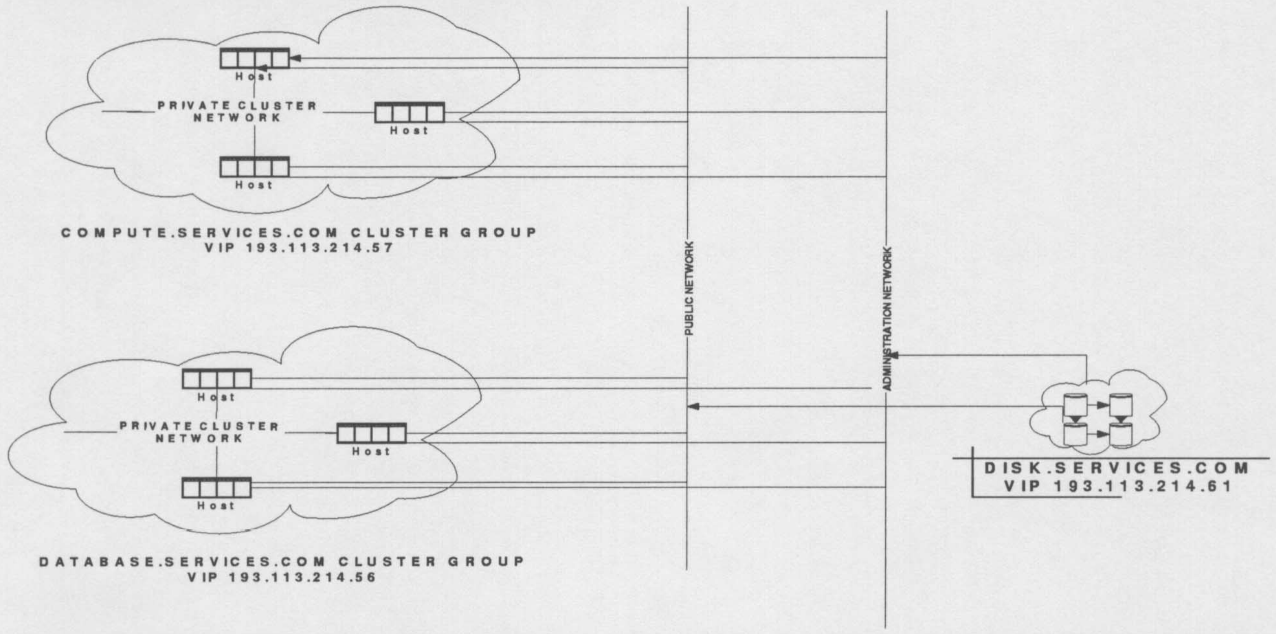

Figure 2. Example of the proposed architecture for biological and health researches. Each application type corresponds to a high-availability cluster group in a fail-over configuration. All hosts within the same cluster group are built in the same way in terms of operating system capabilities. Any host in the cluster can run take any role within the same cluster group. Applications can be distributed or parallel. Addressing is done with Virtual IP addresses and service names to avoid dependencies to physical host IP addresses and names.

3.2 Automated Performance Collection

Performance log collection should be automated, so that system behaviour can be observed accurately. Performance measurement techniques should be orientated towards workgroup aggregation. Measurements should be divided into 5 main groups: 1) Operating system, 2) Network, 3) Disks, 4) Application processes and 5) User processes. Measurements should be kept in a special logs directory and classified first by server name and then by measurement group. All measurements should be recorded in ASCII text files, so they can be later processed by any tool.

The following variables should be observed: 1) I/O rates on disks and network devices, 2) processes per user name, 3) per command name and arguments, 4) per user and command name, 5) per CPU and 6) the match between network packets, port numbers and protocols. Performance monitoring tools should be non-intrusive. For each monitored resource type or workgroup, collected performance logs should be compared against pre-scripted baseline thresholds, in regular intervals (every 10 or 15 minutes). The frequency by which comparisons of this type are made, should be dictated by the nature of the monitored service. For very critical services performance collection should be initiated every 5 minutes and last for 2-3 minutes. Every time a load threshold is exceeded administrators should be notified accordingly.

Different types of measurements should be associated together by matching their creation timestamps. Measurements should be ordered by timestamp and treated as a time series to produce graphical representations of the system performance either as a whole or by component/workgroup. Each file produced by persistent state processes, should be managed as a circular queue, the length of which is configurable. To determine accurately the behaviour of each process, microstate measurements should be used where applicable, as most modern CPUs and linux allow for them. The accuracy of microstate measurements is microsecond resolution and the overhead is sub-microsecond (units are nanoseconds). In this way we can obtain very accurate thread and process accounting. More about microstate accounting can be found in [4].

Baselines must be set based on the hardware configuration of each system and the application type it is meant to be running. These baselines should be determined with the help of hardware and OS experts, application experts and personal observations. Every time a baseline setting is not proven to be correct, it should be adjusted accordingly. This is a common occurrence with newly installed applications. The following measurements should be considered:

- For the operating system: 1) Memory scan rate, page out frequency, page faults and free memory measurements to determine memory shortage, 2) CPU run queue, to detect any processes waiting to be served by any CPU, 3) Overall CPU idle time %, 4) Blocked processes waiting for I/O, 5) Per process CPU and memory utilization, and 6) Disk I/O and throughput in terms of read and write response times.
- For the network: 1) Network interface utilization statistics and errors, 2) Network route utilization, 3) NFS statistics, 4) TCP/IP bandwidth and end-to-end round trip latency measurements, 5) Size of incoming/outgoing network packets and TCP windows, 6) Network connection time to live and 7) Name server response (DNS, NIS, NIS+, LDAP).
- For databases: 1) Time taken for a request to connect to the database, 2) Time taken for the request to be served by the database, 3) Time taken for the database to initialise, 4) Time taken for the database to shutdown, 5) Time taken for the database backup to complete, 6) Per process CPU and memory utilization, 7) Number of users connected to the database and for how long each, 8) Memory allocated at startup, 9) Database checkpoints and 10) Memory per transaction.
- For web servers and application GUIs: 1) Time taken to connect to them, 2) Time taken for the process to come back with the results of the query, 3) Per process CPU and memory utilization, and 4) Number of http/application connections and for how long each.
- For distributed applications: the time taken for a request to be served by the entire application from beginning to end. Every 15 to 30 minutes a “dummy” process can be initiated to run through all application components, simulating a user and measure the total response time, in addition to the “business-as-usual” requests.

4. Results

One of the sites our work was implemented was a financial newspaper of a UK based international customer. Servers included Sun, HP, IBM, Linux and Windows computers. The breakdown of machines and their functions were: 350 database servers, a mixture of Oracle Sybase, Informix, MS Access and DB2 databases, running on Sun, HP, IBM, and Windows servers. Please note that these types of databases and applications are used in many disciplines. The same kind of software is currently being used in the telecommunication, banking and defence industries. In addition, research, biomedical, academic and governmental institutions use them for data-mining and other heavy processing purposes.

About 40 web servers (a mixture of Linux and Sun Solaris machines for Internet users reading the newspaper online) and 200 transaction processing and application servers a mixture of HP, Sun and Linux servers. Services were distributed across these servers. All data resided on local disks. The network was 100 Base/T ethernet for all servers. The majority of data were relational database records in database-specific SQL format. Data feeds were mainly textual ASCII-based streams formatted in a variety of configurations. There were some continuous streams and other data were formatted in single or multi-column configurations. There were also form-based data feeds in HTML or SQL format originating from web servers, Java and database interfaces. Whenever data feeds are in binary format or another format, incompatible with the processing engines, special Perl-based interfaces are engaged to convert them to an appropriate compatible format. Graphical representations are ported directly to databases without additional interface processing. In these cases, database developers have created specialised data objects and schemas for each of the above mentioned database types that are used to import and process them. Images are similarly handled by specialised database tables and schemas.

The newspaper decided to launch an internet based facility to allow users to search through historic financial articles in order to gather mainly investment related information. We were asked to design the new site from the beginning. In addition the customer reported that their current system experienced serious outages and their load balancing techniques were non-existent. On a daily basis, the newspaper received market data feeds that needed to be processed. Financial analysts used services for data-mining, financial projections, financial model evaluations, market data/trend simulations and analytical reports that were given to journalists for publication. It was a high pressure complex environment and downtime had big impacts on service integrity, and business credibility. Delays in financial data processing meant that articles could not be included in the daily publication.

What was happening on a regular basis, was that various application components would stop working altogether and operators did not know where to start looking. Large database jobs scheduled to run overnight would frequently crash databases and calculations would not complete. Human operators tried to resolve operational problems and faults manually. The newspaper would suffer loss of business trust because analysts and researchers could not easily quantify and qualify financial models and analyse market trends. Operators were under immense pressure to resolve operational faults under difficult circumstances and usually during the night and the end of financial periods. In addition web servers crashed very often and users could not access the online newspaper site. To deal with the problems the end-users reported we did the following:

- Redesigned the entire datacentre and grouped hosts together per application and operating system type.
- We had Veritas Cluster server installed and configured for each application group.
- We had load thresholds configured on per server and per application basis, both in the LSF software and the cluster software. These thresholds were determined based on inputs from the hardware manufacturers, application developers/providers, and our own experience and observation.
- We had the customer purchase network based disk storage and move all data to these disks.
- We installed our own custom made software to monitor servers and automatically detect and correct failures. We developed customised scripts to manage and report on incoming market feeds, their status, integrity, validity and processing. We used perl, and Unix shell programs to achieve that. We have developed a full suite of programs that manage servers, clusters and resources, based on intelligent agents and ontologies [2, 3, 8]. Due to space restrictions we cannot analyse our software in this paper. If any information is needed please contact the authors directly.
- We installed an additional administration/backup network to facilitate all administration and maintenance tasks.
- We had the LSF application installed on all servers. It was mainly used for scheduling jobs against databases. Users via the LSF GUI manually selected database servers to submit jobs or submitted them to be processed at a latter time, using either native LSF utilities, or Unix utilities like “cron” or “at” jobs. In addition we automated the use of LSF for internet users who where looking for online articles and had to search a number of databases.

In the figures that follow, we can see the behaviour of the entire system per application type for a total of 12 months (2 six month sets). We have measurements provided by the customer that span for about 1 year; 6 months before any of our work was implemented and 6 months after.

Front-end applications (web servers) had total downtime in hours from all reasons, 6 months prior to work, of 80 hours. After our work was implemented, the following 6 months there was 0 hours downtime (see Figure 3).

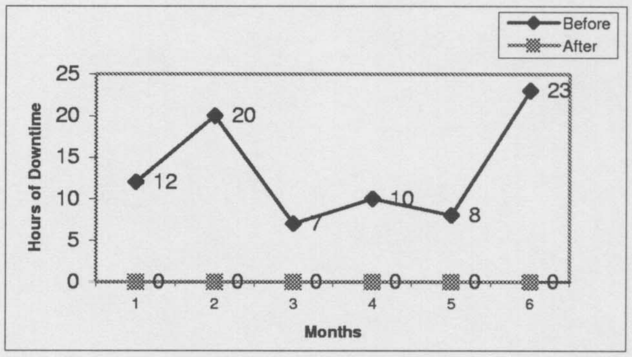

Figure 3. Web server downtime 6 months before any work was done and 6 months after our work. We can see from the “After” series, that downtime was 0. Our configuration ensured that services were always up, although physical machines may had been down. The floating IP addresses would be moved by the cluster software to the next best node automatically if a failure occurred. As data were on commonly accessed disks, no service interruptions occurred.

Back-end applications, like databases, 6 months prior to our work, had a total downtime, of 39 hours, while there was only 1 hour of downtime after our work was implemented (see Figure 4).

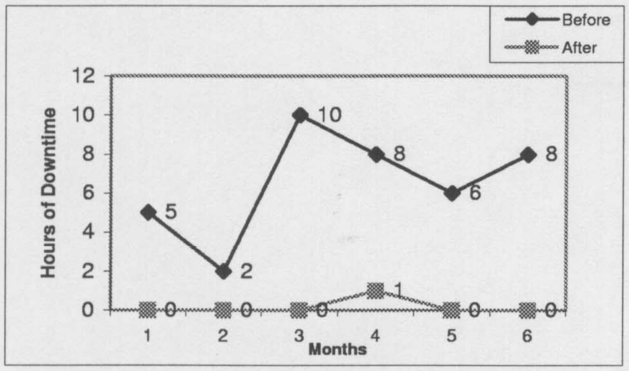

Figure 4. Database downtime in hours per month, 6 months before our work and 6 months after. We can see 1 hour of downtime in month 4, which was caused by a bad data feed to one of the databases.

Combined total downtime for both front and back-end services was 104 hours 6 months prior to our work, as opposed to 1 hour of total downtime after (see Figure 5).

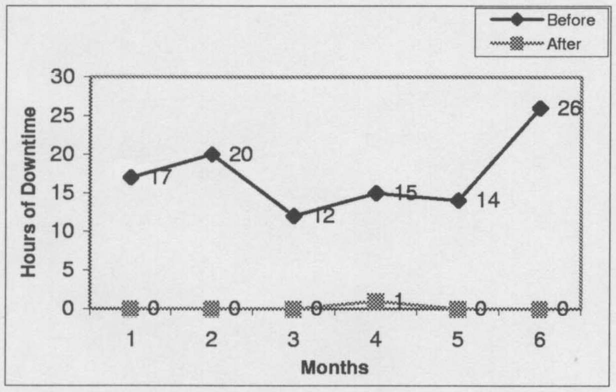

Figure 5. Total downtime for end-customer facing services (Internet users) 6 months before and 6 months after our work. This downtime was caused by web and database unavailability combined. As we can see from the "Before" series, sometimes web servers were down at the same time databases were down. Whenever databases were unavailable however, the web-servers could not be used.

Internal facing customer applications 6 months before any work was done, had 191 hours of total downtime, as opposed to 0 hours of downtime the subsequent 6 months our work was implemented (see Figure 6).

Performance problems were detected and dealt with much faster, as from our automated performance collection techniques we had detailed logs. From these logs, we could understand what may have caused a performance related problem.

As one can see from all the graphs, system uptime for existing customer applications was significantly improved after our work was implemented.

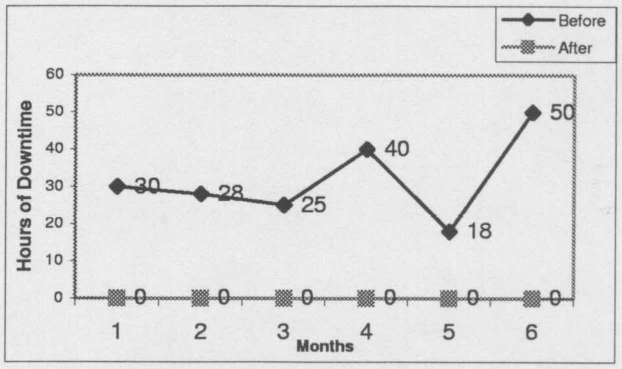

Figure 6. Total downtime for internal customer-facing applications, such as databases used for financial analysis, data feeds etc, i.e. systems used by journalists and analysts on a daily basis. We can see how unstable these services were 6 months before our work was implemented and how stable (0 downtime) they became 6 months after our work.

5. Conclusions and Future Work

Our approach has experimentally improved distributed service availability and uptime in real-time high-pressure environments.

Our main conclusions can be summarized as follows. In the case of complex multi-component applications local application-specific detection/correction mechanisms work much better than generic troubleshooting approaches. High availability cluster software needs to be configured in a specific way to be effective, otherwise it can cause significant problems. Automated error detection and correction techniques improve quality of service as errors are picked up faster than ever before. Methodical and structured performance measurement and collection techniques can help resolve performance related issues and bottlenecks more efficiently and effectively. Administrators can generate timelines of system behaviour and observe similar behavioural patterns. Extended logging of all system ensures that human administrators have comprehensive information about all infrastructure aspects and can narrow down their search options when they do manual troubleshooting.

We make use of existing tested technologies and combine them with application/server specific troubleshooting approaches. We automate maintenance and data related tasks. The latter involves, automation of data feeds, verification, integrity checking and automation of data manipulation procedures. Parts of our approach have also been used in a university environment during an intensive image-processing project. For that project, all pre and post processing image-related activities had been fully automated with the use of Unix shell scripts, Perl, C and C++ programs.

Our work can be used in all environments that have high-availability, performance and dependability requirements. Our building methodology can support parallel and distributed applications equally well with standalone.

Much work remains to be done, so that our automation, error detection and correction techniques are further improved and become more generic. Performance modelling and dynamic troubleshooting of performance-related problems need additional work. Load balancing techniques and threshold setting need further work as well. Our research continues in all these areas, in the hope that we can improve our building methodology and software further.

References

1. Candea George, Cutler James, Fox Armando, Doshi Rushabh, Garg, Priyank, Gowda Rakesh, "Reducing Recovery Time in a Small Recursively Restartable System", Proceedings of the International Conference on Dependable Systems and Networks (DSN-2002), Washington, D.C., June 2002.
2. Corsava Sophia, Getov Vladimir, "Self-Healing Intelligent Infrastructure for Computational Clusters", Proceedings of SHAMAN Workshop at ACM ICS, New York, June 2002.
3. Corsava Sophia, Getov Vladimir, "Intelligent Fault-Tolerant architecture for cluster computing", to appear in Proceedings of PDCN03, IASTED, Innsbruck, Austria, Feb 2003.
4. Cockroft Andrew, "Sun Performance and Tuning", Talk, 2001.
5. Chartier Roger, "Application Architecture: An N-Tier Approach- Part 1", from http://www.15seconds.com/issue/011023.htm,
6. Eckstein Robert, Collier-Brown David, "Using Samba", O'Reilly, 1999.
7. Foster I., Kesselman C., Tuecke S. "The Anatomy of the Grid: Enabling Scalable Virtual Organizations". International J. Supercomputer Applications, 15(3), 2001.
8. Gruber, T.A. "A Translation Approach to Portable Ontology Specifications", 1993.
9. Golding Richard, Borowsky, Elizabeth, "Fault-tolerant replication management in large-scale distributed storage systems", Proceedings 18th IEEE Symposium on Reliable Distributed Systems, 1999.
10. Hellerstein, Joseph, "A comparison of Techniques for Diagnosing Performance Problems in Information Systems: Case Study and Analytic Models", IBM Research Division, 1994.
11. Hellerstein, J, Y. Diao, and S. Parekh, "A First-Principles Approach to Constructing Transfer Functions for Admission Control in Computing Systems", IBM T. J. Watson Research Center, To appear in the Conference on Decision and Control, 2002.
12. Hoschek, Wolfgang, Jean-Martinez Javier,Samar Asad, Stockinger Heinz, Stockinger Kurt, "Data Management in International Data Grid Project", 1st 1EEE, ACM International Workshop on Grid Computing (Grid'2000), Bangalore, India, 17-20 Dec2000.
13. Patterson, D."A new focus for a new century: availability and maintainability >> performance," Keynote speech at USENIX FAST, January 2002.
14. Papoulis, "A. Probability, Random Variables, and Stochastic Processes", 2nd ed. New York: McGraw-Hill, 1984.
15. Veritas Cluster Server, release 1.3.0, Veritas Software Corporation, 2000.
16. Wilkes, John and Keeton, Kimberly, "Automating data dependability", 10th ACM SIGOPS European Workshop, 2002.
17. Wong Kenneth F and Franklin Mark, "Checkpointing in Distributed Computing Systems ", Journal of parallel and distributed computing, vol. 35, 67-75, 1996.
18. http://www.platform.com/products/m/LSF/index.asp
19. http://www.netapps.com
20. http://www.globus.org