Phansalkar and Dani *Journal of Cloud Computing: Advances, Systems and Applications* (2015) 4:13
DOI 10.1186/s13677-015-0038-4

Journal of Cloud Computing
a SpringerOpen Journal

RESEARCH Open Access

Tunable consistency guarantees of selective data consistency model

Shraddha P. Phansalkar[1*] and Ajay R. Dani[2]

Abstract

Tunable consistency guarantees in big data stores help in achieving optimized consistency guarantees with improved performance. Commercial data stores offer tunable consistency guarantees at transaction level where the user specifies the desired level of consistency in terms of number of participating replicas in read and write consensus. Selective data consistency model applies strict consistency to a subset of data objects. The consistency guarantees of data attributes or objects are measured using an application independent metric called consistency index (CI). Our consistency model is predictive and helps in expression of data consistency as a function of known database design parameters, like workload characteristics and number of replicas of the data object. This work extends the causal relationships presented in our earlier work and presents adaptive consistency guarantees of this consistency model. The adaptive consistency guarantees are implemented with a consistency tuner, which probes the consistency index of an observed replicated data object in an online application. The tuner uses statistically derived threshold values of an optimum time gap, which, when padded in a workload stream, guarantees a desired value of consistency index for the observed data object. The tuner thus works like a workload scheduler of the replicated data object and pads only the required time delay between the requests in such a way that desired level of consistency is achieved with minimal effect on performance metrics like response time.

Keywords: Selective consistency; Adaptive consistency; Logistic regression; Workload scheduler; Consistency tuning

Introduction

The consistency, availability and partition tolerance (CAP) theorem [1] states that it is not possible to provide strong consistency and high availability together in presence of network partitioning. Contemporary web and cloud applications achieve high performance and availability by compromising data consistency to an acceptable level.

Optimization of consistency guarantees for better performance is a research area. In replicated databases, data consistency refers to the agreement of the data across multiple copies (replicas) in the system. In a geographically distributed system, a read and update can occur on any replica. When a read operation succeeds an update operation on a data object and cannot read its updated value, it is called a stale read. There is a time delay between an update on one replica and communication of update to other replicas. It is a period of consensus and this period is called unsafe period as most of the reads that occur in this period are likely to be incorrect (highly probable). In our work, consistency guarantees of a data object are measured with a probabilistic metric called Consistency Index (CI). CI enumerates the ratio of the occurrence of correct reads to the total number of reads on a data object in an observed period. In our earlier work, we proposed and proved that selective data consistency model with CI promises better performance as against the consistency models that apply the same consistency constraints to all the data objects.

It is also shown in [2] that CI of any data attribute or object is a predictable consistency metric and can be estimated to a satisfactory level of precision using workload characteristics and the database design parameters. The workload characteristics include the number of reads, updates and the time of their arrival, whereas database replication policy includes characteristics like replication policy and number of replicas of the observed object. In this work, we further extend our earlier work [2] and use CI as a tunable consistency metric. In

* Correspondence: shraddhap@sitpune.edu.in
[1]Symbiosis International University, Pune, India
Full list of author information is available at the end of the article

this work we show that, CI based consistency model can be used to tune the consistency level of data objects or attributes on fly. This feature is important for those applications that exhibit temporal load characteristics. Some web-applications require dynamic adjustment of consistency requirements of data attributes or objects at different levels, as opposed to constant predefined level of consistency for the data attributes or objects in the database. The online auction application requires the bids to be submitted before specified closing time. When an auction starts, a few bids may be submitted, whereas a large number of bids may be submitted just before the closing time of the auction. So the consistency requirements of the bid data object can be less stringent till the auction gains popularity and will be strict towards the closing date and time. This is adaptability of consistency protocol. Our CI based consistency model is capable of providing this kind of adaptive consistency guarantees to the data objects or attributes.

In any real time workload in cloud based or web based applications, a read may be preceded by an update. Predicting correctness of the next read operation is an important milestone for adaptive consistency guarantees. In this work we have used statistical technique of logistic regression to classify next read as correct or incorrect read. This is further enhanced by a neural network classifier. Then the correctness of next read on any data attribute or object can be ensured by introduction of optimum time delay (t_g) and allowing read transaction after this optimum time delay period. The introduction of time delay will prevent the occurrence read transaction in unsafe period. In this work we also derive a relation between the number of replicas (R_p) of a data object or data attributes and minimum optimum time gap (t_g) to be introduced between a read and the preceding update. This time delay would also introduce latencies in the responsiveness of the application affecting its performance. Hence t_g is required to be only long enough to ensure correctness of a read. This leads to the development of a workload scheduler, which introduces only the required threshold time gap (t_o) between an update and a read operation on any data attributes or objects causing minimal adverse effects on the response time. The scheduler is statistically derived and has significant correctness in its prediction. Thus the paper makes following contributions:

- It builds and validates a statistical model that predicts the correctness of next incoming read transaction on a data object or attribute, at any instant of time t (thereby CI) with known number of replicas (R_p) and the time gap (t_g) between the observed read and preceding latest update using logistic regression and radial basis function artificial neural network (RBFNN).
- It estimates minimal threshold time gap (t_o) that must fall between read and update operation on any N- replicated data object which ensures correctness in next read operation.
- Finally it presents the design and implementation of a prototype of a transaction workload scheduler. This scheduler is implemented by consistency tuner for adjusting the consistency level of a selected data object or data attributes to a desired value (adaptive consistency) with minimum effect on the response time.

The rest of the paper is organized as follows. In Section 2, the related work on tunable consistency guarantees is presented. In Section 3, we discuss our CI based consistency model. In Section 4, experimental set up for estimating the correctness of successive read in a workload stream is discussed. The logistic regression and neural network based models for predicting correctness of next read is presented in Section 5 and section 6 respectively. Design and implementation of CI based Consistency Tuner is discussed in Section 7 and section 8 respectively. We present results and discussions in Section 9 and conclude in Section 10.

Related work

In a distributed system, the consistency model has three core dimensions [3]: conflict definition policy, conflict handling policy and conflict resolution policy. In this context, conflict definition policy implies the different definitions proposed for data inconsistencies. Researchers have come up with different strategies of defining access conflict or inconsistency detection for different web-based applications in transactional or non-transactional context.

In [4], a new class of replication systems is proposed called TRAPP (Tradeoff in Replication Precision and Performance), where a query is specified with a precision constant. The constant denotes the maximum acceptable range for imprecision in the answer. In the TACT (Tunable Availability and Consistency Tradeoffs) model [5], the authors develop a *conit-based* continuous consistency model to capture the consistency spectrum using three application-independent metrics viz numerical error, order error, and staleness for the replicated data. The numerical error limits the weight of the writes that can be applied across all replicas before being propagated to a given replica. Order error limits the number of tentative writes that can be outstanding at any one replica, and staleness places a real time bound on the delay of write propagation among replicas. A window based quantitative model that describes consistency as the number of missed updates is proposed in [6]. It is called FRACS (Flexible Replication Architecture for a Consistency Spectrum). This model measures inconsistency in terms of obsoleteness in the data with number of missed updates. Each replica is assigned an update window, which is the maximum number of updates that can be buffered without consensus. The concept of probabilistically bounded staleness (PBS) provides a probability approach using partial quorums to find expected bounds on *staleness* with respect to both versions of the replicas and wall clock time *has been proposed in* [7]. Inconsistency is measured as the number of conflicting updates on data in [8]. An Infrastructure for Detection–based Adaptive Consistency Control in replicated services (IDEA) model [9]–an Internet scale middleware uses an inconsistency detection mechanism of finding version difference in terms of numerical error, order error and staleness. This is an extension to the TACT model [5] for adaptive consistency guarantees with better performance.

Adaptive consistency protocols are also proposed in the distributed shared memory (DSM). A low overhead measurement-based performance metric to determine the most appropriate consistency protocol for each segment in DSM is suggested in [10]. The factors that influence the performance of a consistency protocol, such as network bandwidth, congestion, latency, and topology are considered. DSM models, which adaptively select consistency protocols using heuristic analysis of recent access patterns, have also been proposed. A page-based Lazy Release consistency protocol called Adaptive DSM (ADSM) that constantly and efficiently adapts to the application's sharing patterns is introduced in [11]. Adaptation in ADSM is based on dynamic categorization of the type of sharing experienced by each page. An approach called Harmony [12] uses an intelligent estimation model of stale reads allowing elastically scale up or scale down of the number of replicas involved in read operations. It uses the factors like number of conflicting updates and probability of stale read from [8] for the estimations. File heat based self-adaptive replica consistency strategy [13] allows the system to auto-select an adequate level of consistency by considering the file heat, which is calculated by pattern of file access time and frequency. Upward trend of file heat indicates its criticality and hence system switches to strong consistency guarantees from eventual consistency guarantee. This adaptively reduces the network bandwidth consumption. A work on adaptive leases in web application [14] presents a leases-approach in context of cache consistency in a distributed file system and focusses on finding optimal lease duration to the data object where it would be solely held by a leaser and no modifications to the object would be done without passing of request/reply messages.

Our consistency model like [8] implements consistency as a data characteristic and further extends to selectively apply these constraints to critical data objects for performance. Consistency is measured using CI, which is an application independent metric and can be applied to data of any granularity (data object, tables, collection of tables, row, attribute etc.). CI is well predicted in and its causal relationships with the independent factors like number of reads, updates, number of replicas, and average time gap between a read and preceding update are statistically proved.

This work presents tunable guarantees of our CI based consistency model. Theoretically, tuning any one of the independent determinants would guarantee a desired value of CI. An input workload (R, U) is a random sequence in practice. The number of replicas can be set to a maximum allowable value and is inherently limited by infrastructure. The only dynamically tunable and flexible variable is the time gap between an observed read and a preceding update. Adjusting the value of time gap, it is possible to guarantee the desired level of consistency. Our consistency index based consistency tuner (CICT) implements a workload scheduler, which introduces optimum minimum time delay in the incoming workload in such a way that desired value of CI is achieved for any data attributes or objects for any number of replicas and with minimum impact on performance metrics like response time.

CI based selective data consistency model

We present the selective data consistency model and its inconsistency detection logic here.

A selective data consistency model implies that we apply strict consistency to a subset of data objects which are critical. Thus consistency is treated as a data characteristic. We now present our inconsistency detection logic which finds an incorrect read on a replicated data object. This is followed by definition of an application independent consistency metric called consistency index which is a subject to tunable consistency guarantees.

Unsafe period and incorrect reads

A transaction in a database is a sequence of reads and writes operations on different data objects (or attributes). In a fully replicated distributed system, a read and update operation in two different transactions can occur simultaneously on different replicas of a data object to ensure high availability and low response time. When a replica is updated, the update is conveyed to all other replicas in the system to ensure consistency [15]. The update convergence may be carried immediately (strong consistency with atomic update policy) or with some delay (eventual consistency or deferred write policy). This may lead to stale replicas. Any read on stale replica is an *incorrect read*. The period between an update on a replica and propagation of the updates to all the replicas [t, t + d] is the period of consensus. This period, which has highest probability of incorrect reads, is hereby called *unsafe period*.

Consistency index

A replicated data object undergoes numerous reads and updates in the system. Consistency Index (CI) of a replicated data object is a fraction of correct reads on the object to the total number of reads on all its replicas in the system for an observed period. The formula suggests that CI falls between [0, 1]. Value of CI closer to 1 implies higher consistency whereas value of CI closer to 0 implies lower consistency.

$$CI = \frac{Number\ of\ Correct\ Reads(Rc)}{Total\ number\ of\ Reads(R)} \qquad (1)$$

For a data object when a read is preceded by an update, its correctness depends on:

1. The number of replicas of the data object(R_p)

Higher the number of replicas, larger is the period of consensus [t, t + d]

2. The time gap between the read and the latest preceding update (t_g)

If $t_g < d$, then the read will fall in the unsafe period and can lead to an incorrect read.

Experimental set up for CICT implementation model

We have chosen Transaction Processing Council-C (TPCC) benchmark [16] using Amazon SimpleDB [17] data store of Amazon Web services. It is a popular benchmark for comparing OLTP performance on various software and hardware configurations. It simulates complete computing environment where population of users executes transaction against database. TPCC workloads are a mixture of read only and update intensive transactions for online shopping. It performs five transactions as New Order transaction (which creates new order), Payment transaction (execution of payment), Stock level transaction (reads stock level), Delivery transaction (Delivery of items to the customer) and Order Status transaction (Status of Order placed) on 9 entities. The normalized TPCC schema is shown in the Fig. 1.

Amazon SimpleDB is a highly available and flexible non-relational data store. Developers simply store and query data items via web services requests and Amazon SimpleDB does the rest. It is a No-SQL document oriented data store with a very simple and flexible schema. With weaker consistency guarantees, Amazon SimpleDB is optimized to provide high availability and flexibility, with little or no administrative burden. SimpleDB creates and manages multiple geographically distributed replicas of your data automatically to enable high

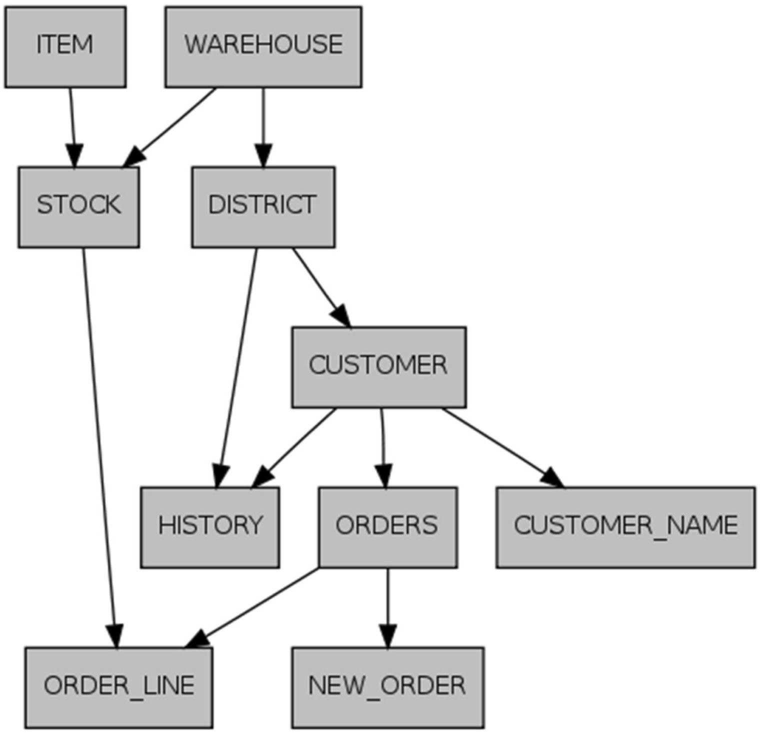


Fig. 1 Normalized TPCC schema

availability and data durability. It organizes data in domains within which you can write data, retrieve data, or run queries. Domains consist of items that are described by attribute name-value pairs. Amazon SimpleDB keeps multiple copies of each domain. A successful write guarantees that all copies of the domain will durably persist.

Amazon SimpleDB supports two read consistency options: *eventually consistent read and consistent read.* An eventually consistent read might not reflect the results of a recently completed write. Consistency across all copies of the data is usually reached within a second; repeating a read after a short time should return the updated data. A consistent read returns a result that reflects all writes that received a successful response prior to the read.

TPCC relational domain mapped on to SimpleDB

In a relational data model, a TPCC object is modelled with a relation and the properties are modelled using attributes. SimpleDB is a No-SQL document oriented data store where *domain* is the unit of data modelling. Also, a SimpleDB domain is a unit of consistency enforcement. A domain is a collection of documents and a document is a collection of fields. A field has value(s) and type. We have mapped a relation in RDBMS to a domain in SimpleDB, a row in RDBMS to a document in SimpleDB, and an attribute in relational model to a field in SimpleDB.

TPCC transaction management

Every normalized table in TPCC is mapped to a separate domain in SimpleDB. TPCC implements five online shopping transactions: New-order transaction, stock-level transaction, order-status transaction, delivery transaction and payment transaction. Every transaction is managed by a separate transaction manager (TM) for scalability. A transaction was subdivided into sub transactions depending on data that they access. Every sub transaction deals with data access or updates of data from different domains to improve throughput with parallelism. In this application, a sub transaction accesses or updates data from a single domain.

Workload on a data object is thus a series of the read and update operations from sub transactions of different transactions with different time stamp. Workload is thus represented by an ordered sequence of read and update operations on the domain. We have then observed the consistency guarantees of the individual domains in isolation. A stock data object in TPCC is accessed by sub-transaction of new-order transaction and sub-transaction of stock level transaction and so is the case with different data objects.

Observed data set structure

Every observation in our data set consists of input predictors and output variable. In our hidden Markov based CI predictive model (HMM-CI) [2], we observe that a read on a data object can be a hidden state and it may be observed as a correct read (1) state or incorrect read (0) state. The output of the read on a data object is predicted with the following parameters:

1. Number of replicas(R_p) of the data object
2. The time gap between the observed read and the latest preceding update request (t_g)

In SimpleDB, a domain is a unit of replication. We have varied the number of replicas (R_p) of stock domain from 1 to 10. We also varied the workload by changing the time gap (t_g) between an incoming observed read and a preceding update in range of 0–2000 milliseconds (ms) in multiples of 500 ms. We have observed the output (O) of a read request by varying R_p and t_g. Thus the triplet (R_p, t_g, O) formed the data set. The predictor was *number of replicas* (R_p) and was treated as a categorical predictor. For a given value of t_g and R_p of a replicated data object, we predicted the correctness of an incoming read request using logistic regression classifier and neural network classifier as described in the next section.

Statistical model of predicting the correctness of a read operation using logistic regression

Logistic regression [18] is widely used to predict a binary response from predictors and is very efficient to predict the outcome of a categorical dependent variable. In our application, the output of a read is categorical, i.e. correct (1) or incorrect (0). Hence the use of logistic regression classifier is suggested.

The regression analysis is run on the collected data set. The Hosmer-Lemeshow test [18] is used for goodness of fit for logistic regression models. R_p is a categorical determinant as the read classifier showed a characteristic behavior in the predicted output for a given value of replica (R_p) with change in time t_g. It is discussed in detail in the next section. The test Chi-Square test statistic with (G-2) degrees of freedom, where G is number of groups, is used to test goodness of fit. In this case, the degree of freedom in Table 1 is 8 (R_p-2). The significance level is close to 0.00. Hence the statistical inferences are significant.

In a logistic regression model fit, the primary measure is pseudo R^2, which are approximations of an indicator of the percentage of variation in the dependent variable explained by the model. Cox and Snell's R^2 depends on the log likelihood for the model with independent variables. Hence its maximum value is less than 1.0 (approximately 0.75 in many cases). It is 0.54 in our results. Nagelkerke's R^2 divides the Cox and Snell's R^2 value by its maximum value and is 0.79 in our results, which implies significant goodness of fit.

Table 1 CI for stock logistic regressive predictor: model summary

Step			−2 Log likelihood	Cox & Snell R Square		Nagelkerke R Square
1			174.892	.592		.790
Hosmer and Lemeshow Test						
Step			Chi-square	Degree of Freedom		Significance
1			54.346	8		.000
Classification Table						
Observed				output		Percentage Correct
				0	1	
Step 1	Output	0		152	18	89.4
		1		18	171	90.5
	Overall Percentage					90.0

In the logistic regression model, the classification table shows the accuracy in the prediction of a sample in a group. The classification percentage was found to be 52 % without the application of the predictor variables where membership of a sample to group is based on the majority (null model). After the application of the predictor variables, the predictive model gave us the classification rate as 90 %, which implies that the classification of a read request into correct read (1) or incorrect read category (0) was accurate to the extent of 90 % after the application of the two predictor variables. This means that the classification model is statistically significant.

Statistical model of CI using radial basis function neural network

The Radial Basis function neural network (RBFNN) [19] has been widely used for prediction and classification tasks. We use the Radial Basis function to model the relationship of correctness of read operation with the same set of independent variables as above. The same data set is used for prediction. The time gap (t_g) is taken as a factor predictor and the number of replicas (R_p) is taken as *covariate* for the reasons to be discussed later. The input data is partitioned as 60 % data as training data set and 40 % testing data set. We have chosen the number of hidden layers as 1. The sum of squared error of test data is 1.953 and the percent incorrect predictions of test data is 8.3 %. The commonly used performance metric for a classifier is the percent correct classification. It is found to be 91.7 % indicating statistically significant results. The *mean-squared error* (MSE) [20] of 0.22 is obtained in the present case. The readings of RBF based output predictor is shown in Table 2. Statistical derivations are done with IBM SPSS [21] tool.

Consistency index based consistency tuner (CICT)

Working principle

It features adaptive consistency guarantees. In [2] we statistically proved that significant variation in CI (dependent variable/outcome) for an observed time period t can be explained by following independent variables with multiple linear regressions:

1. *Number of updates (U) that occur in the observed period* t
2. *Number of reads(R) that occur in the observed period* t
3. *Average time gap (t_g) between a read that follows an update operation that occur in the observed period* t
4. *Number of replicas (Rp) for the observed data object*

Hence CI can also be written as

CI (O, t) $= \alpha + \beta_1 U + \beta_2 R + \beta_3 T + \beta_4 Rp + \varepsilon,$ where ε is error term, O an object and t is the observed time period.

A workload (R, U) is a random and generally unpredictable for an application. The number of replicas is not changed on-fly and generally set to a maximum value in a data store for high availability. The dynamically configurable parameter of time-gap (t_g) is period

Table 2 CI for stock ANN predictive model summary

Training	Sum of Squares Error		9.748	
	Percent Incorrect Predictions		37.2 %	
	Training Time		0:00:00.11	
Testing	Sum of Squares Error		1.953[a]	
	Percent Incorrect Predictions		8.3 %	
Classification				
Sample	Observed	Predicted		
		0	1	Percent Correct
Training	0	11	9	55.0 %
	1	7	16	69.6 %
	Overall Percent	41.9 %	58.1 %	62.8 %
Testing	0	4	0	100.0 %
	1	1	7	87.5 %
	Overall Percent	41.7 %	58.3 %	91.7 %

between a read and the latest preceding update. In the input workload stream, a time gap is introduced between an update and its successive read. We introduce a delay between a succeeding read and its preceding update so that the read will not fall in the unsafe period. This will, however, have an adverse effect on the performance of application in terms of response time. Hence finding minimal time gap which assures correctness of the read operation is the critical issue.

Our consistency model is a CI based selective data consistency model. CI is a data characteristic. Here it may be also noted that the introduction of the time gap would be done on a workload stream on a specific data object. This would be done only when the observed CI of a data object is lesser than desired level of CI, i.e. the observed consistency guarantees of a data object are below the desired ones. Introduction of time delays are avoided for the data objects where observed consistency index is already above the desired value. When the observed value of CI is less than the desired value, a *workload scheduler* pads an appropriate time gap in a workload stream. This would be done in such a way that there is minimum effect on the performance metrics like response time.

As the insertion of time gap is done only when needed, the tuner stabilizes the value of CI of a data object to a desired value guaranteeing performance with desired level of consistency. The working of our Consistency Index based Consistency Tuner (CICT) for adaptive consistency guarantees is discussed in detail in the following sub sections.

Architecture of web based database application with CICT

We discuss the architecture of the CICT based web database application with a block diagram as shown in Fig. 2a and b.

The transaction manager (TM)

An application TM manages the transactions. As discussed, TM is responsible for synchronizing the transactions. In our implementation, it distributes transactions into sub transactions where every sub-transaction accesses a different data object. The atomicity of the sub-transactions is left to the TM. As TM distributes a sub transaction to a data object, a data manager perceives every sub transaction as an independent request to access or update the underlying data object. Thus a data manager works on a stream of read, update requests on the underlying data-object.

The data store

The data store can be a trivial distributed file system or a virtualized No-SQL data store which supports replication. For the deployment of CICT, the data store needs to implement configurable number of replicas. We chose SimpleDB data store in Amazon which is a highly available and flexible non-relational data store. The update policy is write-through i.e. an update to one of the replicas is notified to all the replicas. The succeeding operations are allotted to the replicas with the load allocation policy discussed in the sub-sections. Thus read operations may fall in the unsafe period and these are enumerated as incorrect reads.

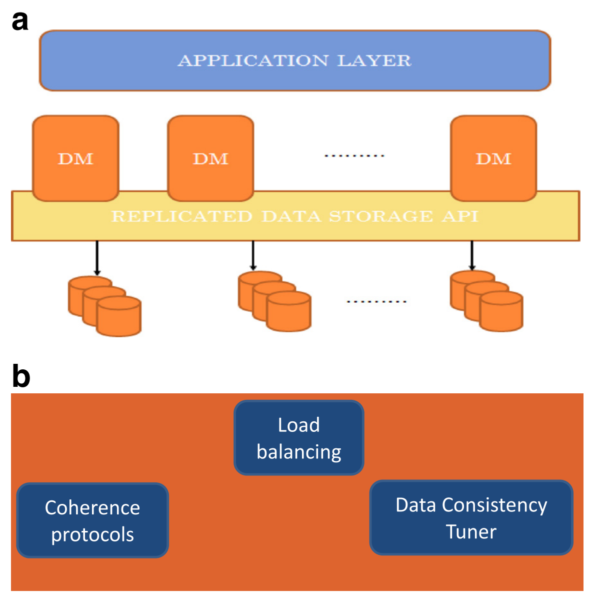


Fig. 2 a Block diagram of architecture of CI based consistency model. **b** Functions of Data manager of CI based consistency Model

The data manager (DM)

The DM carries out the data management operations on the underlying data stores with different data units like domains in SimpleDB [17], HTables in HBase [22] and so on. This layer is a middleware between the raw data store and the application. It performs the three important data management functions, which are otherwise implicitly carried out by databases like RDBMS. DM of a data object controls the number of replicas, location of the replicas. DM can be deployed as a part of application server or can be independent. It works over the replicated data store and the underlying data store is accessed with the APIs provided through DM. Hence our DM is data store dependent.

The CICT works as an integral part of DM. CICT can be very conveniently built on a raw data store as our intelligent DM carries out all the important data management functions. It provides an interface to input the desired values of CI, number of replicas, load balancing policy and replica-update policy to the user. These three important data management functions are represented in Fig. 2b.

1. Replica update policy management implies level of consistency to be implemented. When strong consistency is to be implemented, we use an atomic update policy on the replicas. With eventual consistency policy, we may implement a deferred write policy on the replicas.
2. Load distribution policy implies delegation of a request to one of the replicas. Different strategies like round robin, nearest replica, or heuristic based replica selection policies can be used to improve the performance. Our implementation chooses round robin distribution technique.
3. CICT tunes consistency of the underlying replicated data object to a desired value. The block diagram of CICT is shown in Fig. 3 and discussed below.

We now introduce the architecture of CICT. In the sub sections we elaborate the important building blocks of CICT and they are outlined in Fig. 4.

Threshold time predictor

It is the most important part of CICT which realizes its objectives. As discussed in the previous section, the time gap between read and update request on a replicated data object is dynamically configurable and tunable predictor of CI, which is efficiently employed so that read does not occur in the unsafe period. Thus the value of CI increases. However the increase in the time gap should be just sufficient to lead a correct read and minimally affect the response time. Hence we require finding the minimum value of time gap between an update and a succeeding read request on N-replicated data object which will assure correctness in the read. We refer this minimal time gap as threshold time gap (t_o).

Finding relationship between R_p and t_g to find t_o for a data object O There exists a relationship between the number of replicas (R_p) and time gap between read and update (t_g) on a data object O, which is shown in Fig. 5. For a particular value of R_p, it is observed that when t_g is equal or greater than a threshold value t_o, the output of the read classifier is 1. The value of t_o depends upon the number of replicas (R_p) and are positively related. We captured this relationship with statistical linear regression analysis between R_p and t_o. We, however, note that the underlying data store, replication policy and the size of the object O would also affect the analysis. The analysis holds significant if these factors are held constant. This analysis is different for different data objects. Here we demonstrate how for a stock data object (domain) in TPCC on SimpleDB, we analyzed the relationship. With write-through replication policy, we varied the number of replicas from 1 to 10 and noted t_o. The graph in Fig. 5 shows the value of t_o for values of R_p for stock data object. A statistical linear regression is then run to predict the threshold value (t_o) for any number of replicas (R_p). The coefficient of determination is 0.817 and R^2 is 0.667, which indicates that the regression is

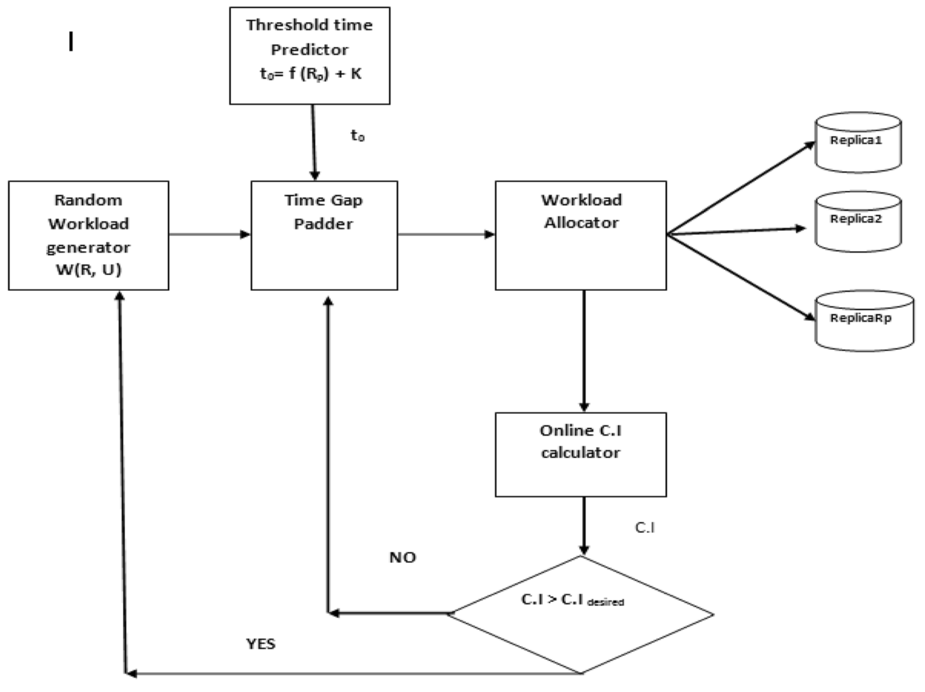


Fig. 3 Steps of CI based Consistency Tuning

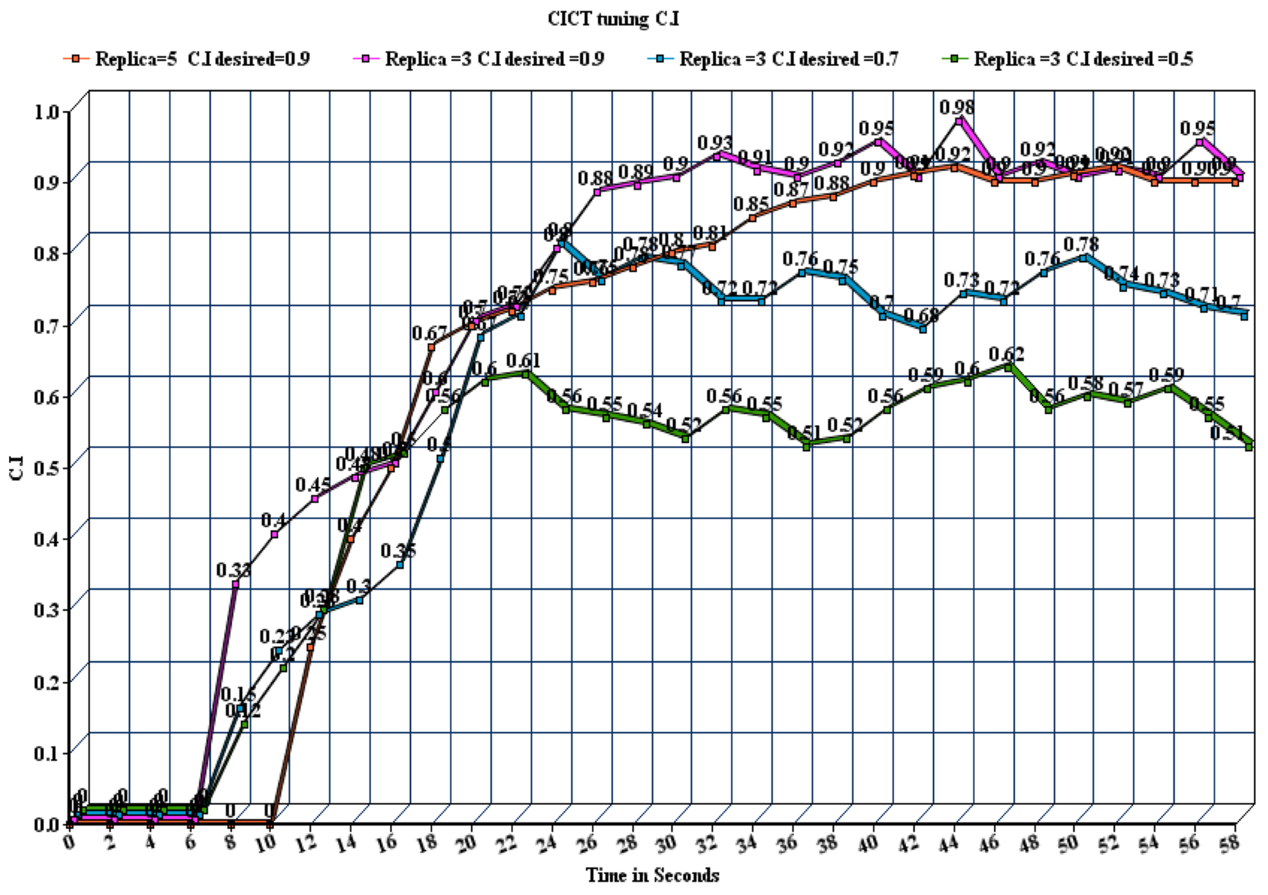


Fig. 4 Stability of CICT for different levels of CI $_{desired}$ and different number of R_p for stock domain in TPCC

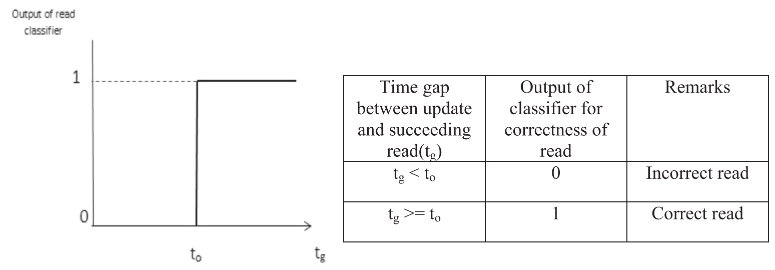
statistically significant. The coefficients in Table 3 indicate the direction as well as the strength of relationship. Also the significance of R_p on t_o is high and validates our hypothesis. The statistical results are shown in Table 3. For CICT of stock data object in TPCC, the threshold was estimated with the coefficient equation given as

$$t_0 = R_p \times 726 + 4723 \tag{2}$$

The threshold predictor of the CICT for stock data object was then used to find the minimum time gap (t_o) that must be inserted between a read and the latest preceding update to assure a correct read (1) on the stock domain. The threshold predictor of CICT of stock domain in TPCC uses Equation 2 to calculate the t_0 for given value of R_p of stock data object. The threshold value (t_0) is then used by the time gap padder to introduce a minimum time gap, which will guarantee the correctness in the read by minimally affecting the performance.

Time gap padder

CICT probes the value of CI periodically. The periodicity can be after every operation or periodically after



Time gap between update and succeeding read(t_g)	Output of classifier for correctness of read	Remarks
$t_g < t_o$	0	Incorrect read
$t_g >= t_o$	1	Correct read

Fig. 5 Relationship between R_p and t_g to find t_o for a data object O

Table 3 Linear Regression results between R_p and t_0 for stock

Model	R	R Square	Adjusted R Square	Change Statistics		
				R Square Change	F Change	Sig. F Change
1	.817	.667	.666	.667	756.658	.000

Coefficients′

Model	Unstandardized Coefficients		Standardized Coefficients	t	Sig.	95.0 % Confidence Interval for B	
	B	Std. Error	Beta			Lower Bound	Upper Bound
(Constant)	4723.333	151.707		31.135	.000	4425.038	5021.628
Replicas	726.553	26.413	.817	27.507	.000	674.618	778.488

some time gap. We chose a time gap of 5 s. If the observed CI is less than the desired value of CI, we trigger the time gap padder. The padder inputs the threshold value for the given database design (t_0) from threshold predictor and then pads the input workload with t_0 so that the read follows the update with t_0 delay to ensure the correctness of the read operation. Time gap padding is operationally implemented by delaying the read operation by t_0. Thus the reads are refrained from the unsafe period and outputted as correct reads. This results in improvement in the value of CI. If the CI is above the desired value, the time gap padder is bypassed and the workload is directed to the replica allocator. Thus the performance is maintained. Thus we get a stable desired value of CI with minimal compromise on performance.

Workload allocator

The workload allocator implements a replica selection policy. The replica selection policy can be based on one of the strategies like *round robin policy* which is simple to implement. The other strategies include the *least busy replica policy,* which promises a good response time with an overhead of maintaining the status information and the *nearest replica policy,* where the geographical distance between the requestor and the replica should be tracked and nearest replica should reply. Round robin load balancing policy is chosen for our implementation.

We have implemented CICT on TPCC application with SimpleDB data store. CICT is implemented in the data manager of every data object.

Implementation of CICT for TPCC

We have implemented CICT on the TPCC workload benchmark [16] which models an online shopping application. The original relational database of TPCC was migrated to Amazon SimpleDB [17] as already described in the previous section. We implemented five (5) transactions as distributed sub-transactions, where every sub-transaction accessed different data object. A data manager thereby gets a stream of sub transactions which are dedicated on the underlying data object.

The input workload to a CICT is a series of read and update operations. In case of a real time web application, this input is a random sequence of read and update operations separated with random time gap. As we had to control the input workload, the sub transactions on a data object are modelled using a multithreaded, randomly generated sequence of read and updates. Thus the workload was randomized with respect to the number of reads and updates and their occurrence. The database design in terms of number of replicas was statically fed to the tuner and varied for observations. The desired value of CI ($CI_{desired}$) was fed and varied for different observation sequence. The threshold–time predictor uses the statistical linear regression between the time gap and number of replicas as discussed in 7.2.4. The time gap padder then pads the minimum time gap between an update and a succeeding read so that it results in correct read. This results in increase in the value of CI at the cost of response time. When CI reaches the user defined value, the time delay is not introduced until the next poll.

Results and discussion

The graph in Fig. 4 shows the stability of the CICT for the object *stock* (*stock* domain) in the TPCC schema. We have varied the desired value of CI as well as the number of replicas (R_p) for the stock domain. For each pair, we have observed that the tuner assures a stable value of CI which is close to its desired value. The graph in Fig. 6 shows that the time gap (t_o) required to introduce between an update and read increases with the increase in the number of replicas. This is also obvious with the statistically derived Equation 2. This implies that response time is higher with higher number of replicas and high values of desired CI. This is shown in Fig. 7.

New-order transaction accesses five different data objects. Response time of the transaction is the summation of the average response time of all its sub-transactions. We adopt our selective consistency model where critical data objects are strictly observed. The tuner applies the desired value of consistency index to selected data object. It allows the user to tune consistency level of different data objects at different levels and achieve the

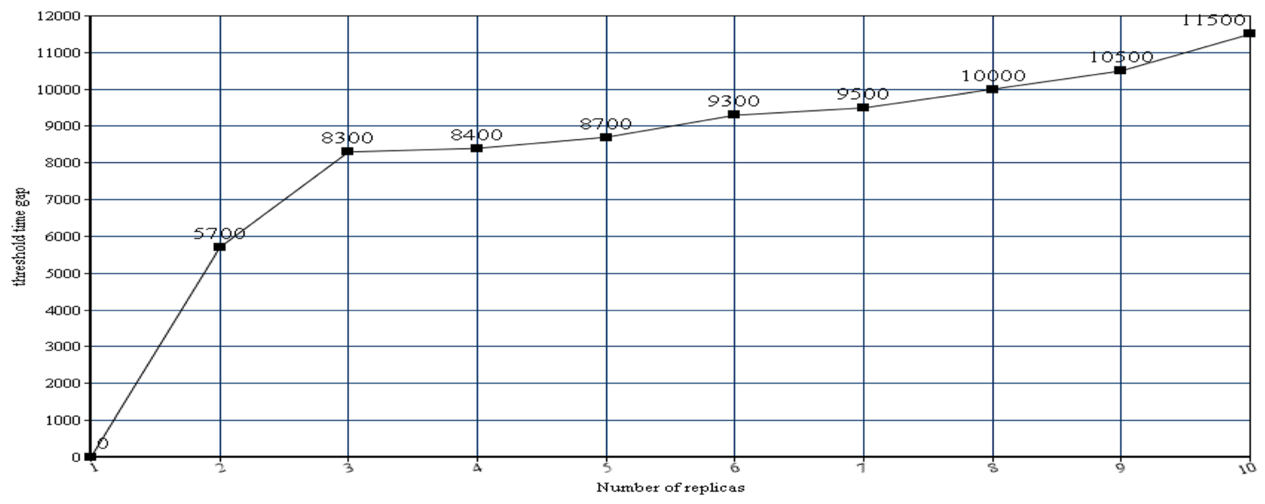


Fig. 6 Relationship between threshold time gap (t_o) and number of replicas (R_p) for stock domain in TPCC

performance goals of response time with stable and desired consistency guarantees of critical data objects. With threshold time gap, it works like a workload scheduler for any workload stream on a data object to achieve desired value of consistency index with minimal adverse effects on response time.

Conclusion

The work demonstrates adaptive guarantees of selective data consistency model. It demonstrates consistency as a data characteristic. It allows us to be selective to the data. The inconsistency detection logic is simple and flexible and guarantees a predictive and tunable consistency metric with CI. CICT works on the principle of delaying the conflicting read operations to refrain it from falling in the unsafe period. The time delay is a statistically derived value that allows us to stabilize the consistency guarantees of a data object to a desired value. The time gaps are interspersed between the input workload only when the consistency guarantees are lower than the desired level. Thus CICT causes minimal degradation in the performance. Thus CICT can be efficiently exploited as workload scheduler that tunes value of CI of a data object to a desired value for any number of replicas and any

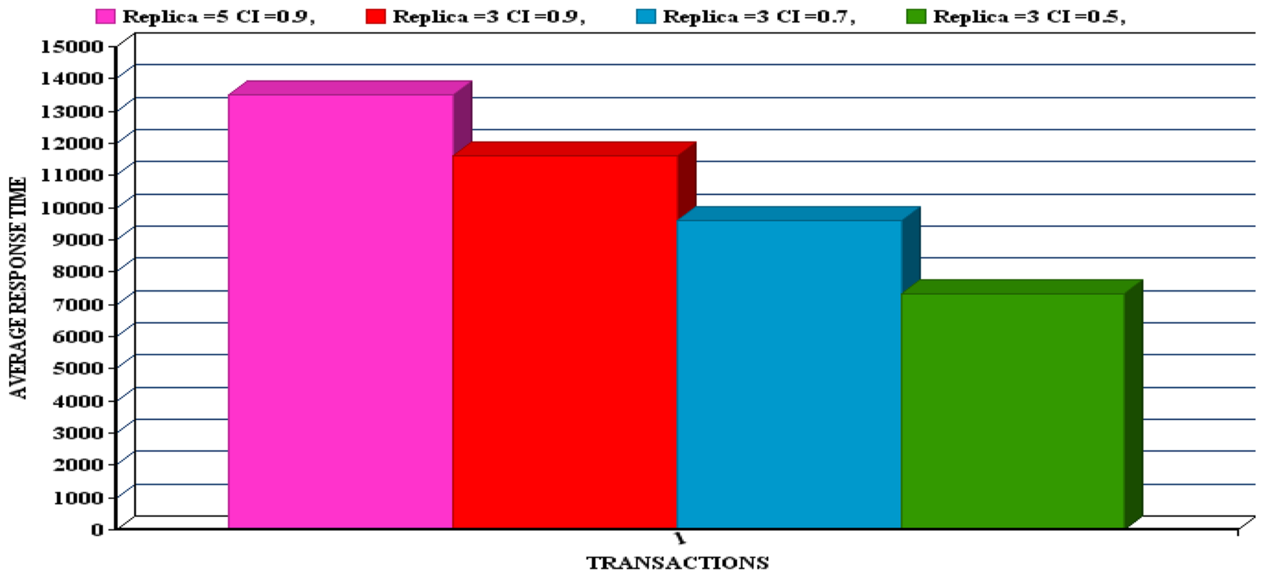


Fig. 7 Average response time of a sub-transaction with different number of replicas and CI $_{desired}$ values for stock domain in TPCC

random workload to achieve minimal degradation in the application performance. The work however assumes default operation strategies for transaction division (into sub-transactions), replica update policy and load balancing strategies. The effects of each of these strategies on the performance can be explored in our future work.

Abbreviations

CAP: Consistency availability and partition tolerance; NO-SQL: Not only SQL; RBF-ANN: Radial basis function artificial neural network; CI: Consistency index; $CI_{desired}$: Consistency index (desired value); CICT: Consistency index based consistency tuner; R: Coefficient of determination; R_p: Number of replicas; t_0: Threshold; t_g: Time gap; TPCC: Transaction processing performance council; OLTP: Online transaction processing; MSE: Mean squared error.

Competing interests

The authors declare that they have no competing interests.

Authors' contributions

SPP has been the key contributor of the work. The conception of the work and the design of the system is the contribution of SPP. SPP has been involved in the design of the tuner. She has been the contributor in the sequence alignment of the manuscript. ARD has contributed with the conception of the logistic regressive model. He has been involved in drafting the manuscript or revising it critically for important intellectual content. Both authors read and approved the final manuscript.

Authors' information

Shraddha Phansalkar received her M.E (Computer science) with gold rank from Mumbai University and is specialized in Distributed Systems. She works as Asst. Professor in Symbiosis Institute of Technology under Symbiosis International University and is a research student at the same university. Dr A. R. Dani holds an M.Tech in Computer Science from IIT, Kharagpur and Ph.D. in Computer Science from Hyderabad Central University (UOH-AP, India). He has around 20 years of industry experience and worked with Reserve Bank of India and Institute for Development Research in Banking Technology (IDRBT). He has worked on critical projects of national Payment Systems.

Acknowledgements

We sincerely thank Dr. Arundhati Warke who helped us with the theories of the statistical methods like regression.

Author details

[1]Symbiosis International University, Pune, India. [2]G. H. Raisoni Institute of Engineering and Technology, Pune, India.

Received: 23 December 2014 Accepted: 3 June 2015
Published online: 10 June 2015

References

1. Brewer E (2000) Towards robust distributed systems'. In: Proceedings of the 19th ACM symposium on principles of distributed computing., pp 7–10
2. Phansalkar S, Dani A (2015) Predictive models for consistency index of a data object in a replicated distributed database System'. WSEAS Transactions on Computers 14:395–401, Art. #40
3. Crooks N, Bacon J, Hand S (2013) War of the Worlds: Branch Consistency in Distributed Systems, Report presented at University of Cambridge, Computer Laboratory, UK, April.
4. Olston C, Widom J (2000) Offering a precision-performance trade-off for aggregation queries over replicated data'. In: Proceedings of the 26th international conference on very large databases., pp 144–155
5. Haifeng Y, Vahdat A (2002), 'Design and Evaluation of a Conit-Based Continuous Consistency Model for Replicated Services', ACM Transactions on Computer Systems, Vol. 20, No. 3, pp. 239-282.
6. Zhang C, Zhang Z (2003) 'Trading Replication Consistency for Performance and Availability: an Adaptive Approach', Proceedings of the 23rd International Conference on Distributed Computing Systems, pp.687-695.
7. Bailis P, Venkataraman S, Franklin MJ, Hellerstein JM, Stoica I (2012) Probabilistically bounded staleness for practical partial Quorums'. Proc VLDB Endow 5(8):776–787
8. Kraska T, Hentschel M, Alonso G, Kossmann D (2009) Consistency rationing in the cloud: pay only when it matters. Proc of VLDB 2:253–264
9. Yijun L, Ying L, Hong J (2008) Adaptive consistency guarantees for large-scale replicated services. In: Proceedings of the international conference on networking, architecture, and storage., pp 89–96. doi:10.1109/NAS.2008.64
10. Monnerat LR, Bianchini R (1998) Efficiently adapting to sharing patterns in software DSMs'. In: Proc. 4th IEEE symposium on high-performance computer architecture., pp 289–299
11. Christopher D, James G (2000) A measurement-based algorithm for mapping consistency protocols to shared Data'. In: The second international workshop on software distributed shared memory (in conjunction with the international conference of supercomputing)
12. Chihoub HE, Ibrahim S, Antoniu G, Perez MS (2012) Harmony: towards automated self-adaptive consistency in cloud Storage'. In: IEEE International Conference In Cluster Computing (CLUSTER)., pp 293–301
13. Zhou Z, Chen S, Ren T, Wu T (2014) File heat-based self-adaptive replica consistency strategy for cloud Storage'. J Computers 9(8):1928–1933
14. Duvvuri V, Shenoy P, Tewari R (2000) Adaptive leases: a strong consistency mechanism for the world wide web'. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM) 2:834–843
15. Tanenbaum A, Van Steen M (2007) Distributed systems, 2nd edn. Pearson Prentice Hall, NJ, USA
16. Transaction Processing Performance Council, TPC benchmark C standard specification, revision5.11, http://www.tpc.org/tpcc/ (accessed Oct 2014).
17. Amazon SimpleDB Documentation, http://aws.amazon.com/simpledb/ (accessed Oct 2014)
18. Hosmer DW Jr, Lemeshow S (2004) Applied logistic regression'. John Wiley & Sons, Hoboken, New Jersey
19. Schalkoff RJ (1997) Artificial neural networks'. McGraw-Hill, New York
20. Christian W (2007) Handbook on statistical distributions for experimentalists. Internal Report, University of Stockholm, pp 69–143
21. IBM SPSS version 20: Reference Guide for IBM SPSS Statistics, http://www.ibm.com/software/analytics/spss/products/modeler/ (accessed Oct 2014)
22. George L (2011) HBase: the definitive guide. O'Reilly Media, Inc, Sebastopol, CA