Reducing Transaction Abort Rate of Epidemic Algorithm in Replicated Databases

Huaizhong Lin[1,*], Zengwei Zheng[1,2], and Chun Chen[1]

[1] College of Computer Science, Zhejiang University, 310027 Hangzhou, China
{linhz, zhengzw, chenc}@zju.edu.cn
[2] City College, Zhejiang University, 310015 Hangzhou, China
zhengzw@zucc.edu.cn

Abstract. Easy to deploy, robust, and highly resilient to failures, epidemic algorithms are a potentially effective mechanism for propagating information in large peer-to-peer systems deployed on Internet or ad hoc networks. In the paper, we explore the epidemic algorithms used for transaction processing in replicated databases that reside in weakly connected environments. We concentrate on the transaction commit voting process of the epidemic algorithms and suggest a new voting method, which takes an optimistic approach in conflict reconciliation. The optimistic voting protocol decreases abort rate and improves average response time of transactions.

1 Introduction

In recent years, the wireless communication and wide area network technologies, especially Internet, evolve rapidly. Weakly connected environments, which are characterized by low bandwidth, excessive latency, instability of connection, and constant disconnection, are used more and more frequently. Data replication is the common approach to improve system performance and availability. But due to the massive communication overhead in weakly connected environments, eager replication may bring about unacceptable number of failed or blocked transactions, and result in dramatic drop of system performance [1,2].

Epidemic algorithms [3], which mimic the spread of a contagious disease, have recently gained popularity as a potentially effective solution for disseminating information in large-scale systems, particularly P2P systems deployed on Internet or ad hoc networks. In addition to their inherent scalability, they are easy to deploy, robust, and resilient to failure. It is possible to adjust the parameters of an epidemic algorithm to achieve high reliability despite process crashes and disconnections, packet losses, and a dynamic network topology.

Epidemic algorithms can be used for managing replicated data [4-9]. In an epidemic approach, sites perform update operations locally and communicate peer-to-peer in a lazy manner to propagate updates. Transactional consistency is achieved by

* Supported by the Natural Science Fundation of Zhejiang Province, China (Grant no. M603230) and the Research Fund for Doctoral Program of Higher Education from Ministry of Education, China (Grant no. 20020335020).

V.S. Sunderam et al. (Eds.): ICCS 2005, LNCS 3514, pp. 552 – 559, 2005.
© Springer-Verlag Berlin Heidelberg 2005

decentralized conflict detect and reconciliation. Sites communicate in a way that maintains the causal order of updates and the communication can pass through one or more intermediate sites. Therefore, the epidemic model provides an environment that is tolerant of communication failures and doesn't require continuous connection between sites. Epidemic model is suitable for transaction processing of replication systems in weakly connected environments.

Several protocols have been proposed for implementing epidemic model in replicated databases, like ROWA (Read-One Write-All) protocol [4], quorum protocol [5], voting protocol [7,8], etc. In this paper, we describe the optimistic voting protocol, which introduces *condition* and *order* vote in the election process in transaction commitment. *Condition* vote postpones the final decisions on conflicting transactions and therefore improves the chances for transactions to get *yes* vote. *Order* vote prescribes the commit order of transactions that have read-write and write-write conflicts and eliminates transaction aborts due to these kinds of data conflicts. Optimistic voting protocol reduces abort rate and improves average response time of transactions when compared to other protocols.

The rest of the paper is organized as follows. In section 2, we develop the necessary background and introduce the epidemic model used in replicated databases. In section 3, we describe the optimistic voting protocol. In section 4, we perform the performance evaluation. We conclude the paper in section 5.

2 Epidemic Model

We consider a distributed system consisting of n sites labeled $S_1,S_2,\ldots,S_n$ and data items replicated fully or partially at all sites. Epidemic model assumes a fail-stop model of site failures and an unreliable communication medium. Sites communicate each other through messages passing in a pair-wise manner. Messages can arrive in any order, take an unbounded amount of time to arrive, or may be lost entirely, however, messages will not arrive corrupted. For this reason, timeout is not used in the protocols to detect conflicts and deadlocks.

Epidemic model is based on the causal delivery of log records where each record corresponds to one transaction instead of one operation. An event model [5] is used to describe the system execution, $(E, \rightarrow)$, where E is a set of transaction events and $\rightarrow$ is the happened-before relation which is a partial order on all events in E. The partial order $\rightarrow$ satisfies the following two conditions:

(1) Events occurring at the same site are totally ordered;
(2) If e is a sending event and f is the corresponding receiving event, then $e \rightarrow f$.

Vector clocks are used to ensure the property that if two events are causally ordered, their effects should be applied in that order at all sites. Each site S_k keeps a two-dimension time-table, which corresponds to S_k's most recent knowledge of the vector clocks at all sites. Upon communication, S_k sends a message including its own time-table and all records that receiving site hasn't received. Then the receiving site processes the events according to causal order and incorporates the time-table in an atomic step to reflect the new information from S_k.

The site S_k determines the records that receiving site S_j hasn't received according the following predicate [5] :

$HasRecvd(T_k, t, S_j) \equiv T_k[j, Site(t)] \geq Time(t)$

Where t is an event, Site(t) is the site at which t occurred, and Time(t) is the local time at Site(t) when t occurred.

Upon completion of operations, a read-only transaction can be committed locally whereas an update transaction pre-commits and becomes a candidate. The read set, write set, and the update values of the candidate are recorded in log. Then sites exchange their respective log records to detect global conflicts and propagate values written by the transaction. A candidate is voted on and is eventually either committed (if it wins a plurality of the total system votes) or aborted.

When a transaction pre-commits, it is attached with a global distinct timestamp denoted by (local_ts, site_index), which is composed of a local timestamp and a distinct site index. Formally, we define a total order < on timestamps as follows. Suppose two timestamps $ts(T_1)=(local\_ts_1, site\_index_1)$ and $ts(T_2)=(local\_ts_2, site\_index_2)$, then $ts(T_1)<ts(T_2)$ if and only if:

(1) $local\_ts_1<local\_ts_2$, or
(2) $local\_ts_1=local\_ts_2$ and $site\_index_1<site\_index_2$.

The information of local timestamp is piggybacked in the usual epidemic messages and a site adjusts its local timestamp as follows [10]: when site A receives a message from site B, it advances its local timestamp to max{ $local\_ts_A$, the $local\_ts_B$ carried by message}. If there are no communications between sites, their local timestamps will drift apart. But this doesn't matter since, in the absence of such communications, there is no need for synchronization in the first place and the drift will not affect the correctness of the protocol.

3 Optimistic Voting Protocol

3.1 Condition and Order Vote

Suppose two conflicting transactions T_i and T_j are issued by two sites concurrently. To maintain serializability, previous epidemic protocols consider that there is only one transaction can be committed and each site can only cast *yes* vote to one transaction in election, for example T_i. In optimistic voting protocol, to increase the chances to get *yes* vote for transaction T_j, sites can cast *condition* vote on it (whereas it is cast *no* vote in quorum or voting protocols). The *condition* vote on T_j can be transformed to *yes* vote if T_i is aborted. The use of *condition* vote postpones the final vote decision on transactions.

Definition 1. When voting on transaction T, suppose $C=\{T_1,\ldots,T_p\}$ is the set in which each transaction conflicts with T, the *condition* vote *cond*(C) means that it can be transformed to *yes* vote in case each transaction in C is aborted, otherwise to *no* vote.

The transform rules of *condition* vote are as follows:

(1) If $\exists T_i \in C$, T_i has been aborted, then $cond(C) \rightarrow cond(C\text{-}T_i)$;
(2) If $\forall T_i \in C$, T_i has been aborted, then $cond(C) \rightarrow yes$;
(3) If $\exists T_i \in C$, T_i has been committed, then $cond(C) \rightarrow no$.

For two transactions T_i and T_j that only have read-write and write-write conflicts, if the correct order can be preserved at all sites, e.g. T_i is committed before T_j, then the two conflicting transactions T_i and T_j can all be committed maintaining consistency. *Order* vote prescribes the commit order of these kinds of conflicting transactions. Additionally, it is easily observed that *condition* and *order* vote can coexist on one transaction T.

Definition 2. When voting on transaction T, suppose $C=\{T_1,\ldots,T_p\}$ is the set in which each transaction has only read-write and write-write conflicts with T, the *order* vote *order*(C) means that it can be transformed to *yes* vote when all transactions in C have been committed or aborted at one site.

The transform rule of *order* vote is as follows:

If at one site, $\forall T_i \in C$, T_i has been committed or aborted, then *order*(C) → *yes*.

Each site S_k maintains a list of candidates by the receiving order. Let $list_k$ denote the candidate set in which the vote on each transaction by the site is not *no* vote. When S_k receives a new candidate T, it votes on T according to the following rules. For convenience of description, Let

cond_set=$\{ T_i \mid T_i \in list_k$, wr_conflict($T_i$,T) is true $\}$,

order_set=$\{ T_i \mid T_i \in list_k$, rw_conflict($T_i$,T) or ww_conflict($T_i$,T) is true, and wr_conflict(T_i,T) is false $\}$.

(1) If $\exists x \in$ ReadSet(T), ReadVN(T,x)<CurrVN(S_k,x), it means that the value read by T has been overwritten, then vote *no*;
(2) If cond_set=$\varnothing$ and order_set=$\varnothing$, it means that there are no transactions in $list_k$ that have conflict with T, then vote *yes*;
(3) If $\exists T_i \in$ cond_set$\cup$order_set, ts(T_i)>ts(T), then vote *no*;
(4) If $\forall T_i \in$ cond_set$\cup$order_set, ts(T_i)<ts(T), then vote *cond*(cond_set) + *order*(order_set). The '+' denotes that the vote is transformed to *yes* vote if and only if both the *condition* and *order* vote are transformed to *yes*, otherwise to *no* vote.

The correctness proof of optimistic voting can be found in [9].

The votes collected in optimistic voting protocol can be viewed as optimistic quorum. The optimistic quorum differs from ordinary quorum in replicated databases in that the quorum is conditional and can only be transformed to really quorum based on the results of other transactions. This optimistic quorum increases the chance for a transaction to win a majority of sites, thus reducing the transaction abort rate.

3.2 An Example

We explain the optimistic voting protocol with an example. Suppose three transactions T_1, T_2 and T_3 (ts(T_1)<ts(T_2)<ts(T_3)).

Fig. 1 shows the voting process of voting protocol. Because three transactions have data conflict with each other, only one transaction can be committed. Fig. 2 shows optimistic voting process, which avoid the abort of T_3 by use of *order* vote. From the figures, we observe that T_2 in optimistic voting can be committed earlier than in

voting protocol by use of *order* vote. For clarity, we omit some unimportant information exchanges in Fig. 1 and Fig. 2.

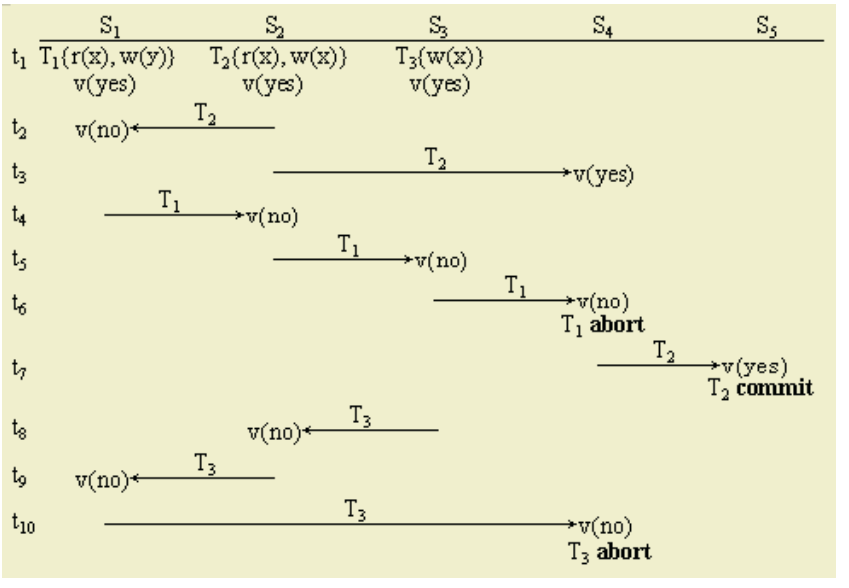

Fig. 1. Voting protocol

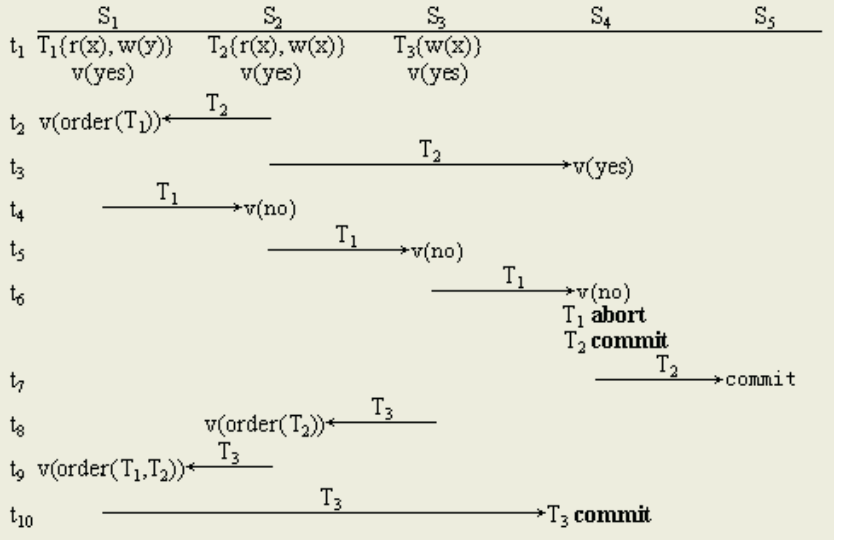

Fig. 2. Optimistic voting protocol

4 Performance Evaluation

We perform experiments to show performance improvement attained by optimistic voting (OV) protocol. Additionally, we investigate two representative epidemic replication schemes from the literature, ROWA protocol [4] and voting protocol [7] (quorum protocol [5] is similar to voting protocol). The evaluations are done at 10 desktops connected via a 10Mbps Ethernet network.

The simulation assumes that data items are fully replicated at all sites and tickets are uniformly distributed among sites. Each site generates transactions randomly according to a global transaction generation rate. Data items are accessed uniformly by transactions. Each site periodically initiates a synchronization session with a given synchronization interval by sending a pull request to another randomly selected site.

Since we focus on the transaction abort rate and commit delay of different protocols, we don't model any read-only transactions. Each transaction read 5-10 data items and write 5 data items that are in the read set, so there are no blind writes. The main parameters and settings used in the experiments are summarized in Table 1.

Table 1. Experimental parameters

Parameters	Descriptions	Values
N	Site number	10
Sync. interval	Average synchronization interval	1~5s
Trans. rate	Average generation rate of update transactions	0.2~20/s
Data items	Total data item number	500

Fig.3 illustrates the transaction abort rate of three protocols for various values of transaction generation rate. From the figures, it is obvious that optimistic voting protocol outperforms the other two protocols.

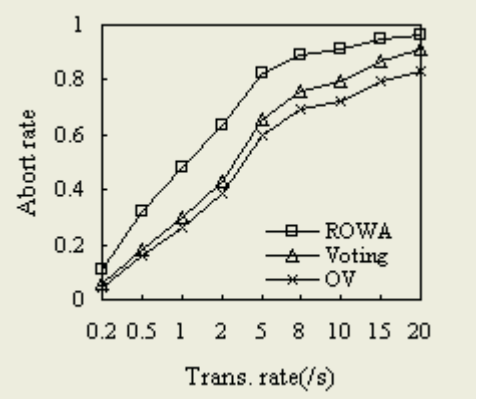

Fig. 3. Abort rate vs. transaction generation rate (Synchronization interval=1.0s)

In optimistic voting protocol presented above, the *condition* and *order* vote of a transaction is dependent on other transactions. This dependency relation in optimistic voting is one-way, i.e. a transaction can only depend on transactions that have smaller global timestamp than it (noted as protocol A). The one-way dependency ensures that there are no cycles among transactions and therefore no global deadlocks. This one-way dependency can be converted to depending on transactions that have larger global timestamp (noted as protocol B). We explore the impacts on performance of different dependency direction by experiments. Fig.4 and Fig.5 illustrate the transaction abort rate of protocol A, protocol B, and voting protocol for various values of transaction generation rate and synchronization interval. From the figures, it is obvious that protocol A is better than protocol B with average 5.4% performance gain. It is the natural direction for a transaction to depend on other transactions with smaller global timestamp. The transaction with smaller timestamp has been stay in the system for a longer time span and will be committed or rollbacked much earlier, which makes the transform of *condition* and *order* vote more quickly, thus reduce the delay of a transaction in the system. Additionally, we can notice that both protocol A and B outperforms the voting protocol.

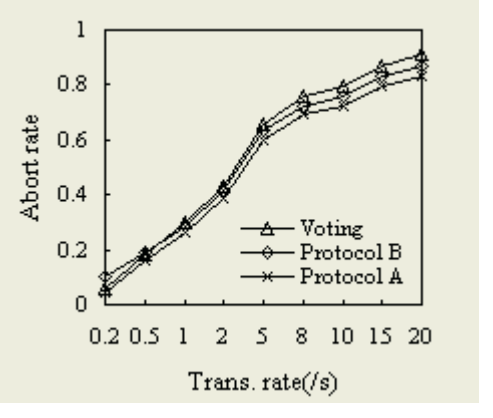

Fig. 4. Abort rate vs. transaction generation rate (Synchronization interval=1.0s)

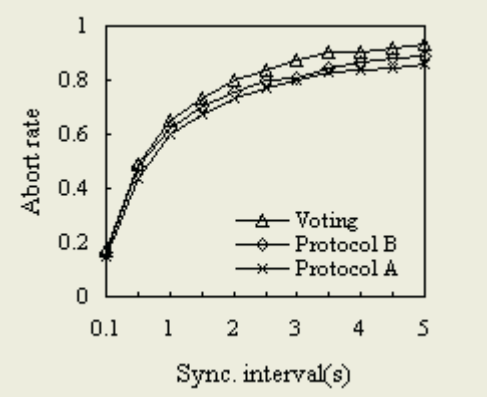

Fig. 5. Abort rate vs. synchronization interval (Transaction generation rate=5.0/s)

5 Conclusion

Epidemic replication schemes are used extensively in transaction processing in weakly connected environments. Some continuously connected systems also use epidemic model to improve system efficiency. The optimistic voting protocol presented in this paper improves system performance in epidemic model and is of high practical values.

References

1. J. Gray, P. Helland, P. O'Neil, and D. Shasha. The dangers of replication and a solution. In Proceedings of ACM SIGMOD International Conference on Management of Data. Montreal, Canada, 1996. 173-182.
2. T. Anderson, Y. Breitbart, H. F. Korth, and A. Wool. Replication, consistency, and practicality: are these mutually exclusive. In Proceedings of ACM SIGMOD International Conference on the Management of Data. Seattle, Washington, 1998. 484-495.
3. Patrick T. Eugster, Rachid Guerraoui, Anne-Marie Kermarrec, and Laurent Massoulié. Epidemic Information Dissemination in Distributed Systems. IEEE Computer, 2004, (5): 60-67.
4. D. Agrawal, A. El Abbadi, and R. Steinke. Epidemic algorithms in replicated databases. In Proceedings of 16th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems. Tucson, Arizona, 1997. 161-172.
5. J. Holliday, R. Steinke, D. Agrawal, and A. El Abbadi. Epidemic quorums for managing replicated data. In Proceedings of 19th IEEE International Performance, Computing, and Communications Conference. Phoenix, Arizona, 2000. 93-100.
6. K. Petersen, M. J. Spreitzer, D. B. Terry, M. M. Theimer, and A. J. Demers. Flexible update propagation for weakly consistent replication. In Proceedings of 16th ACM Symposium on Operating System Principles. St. Malo, France, 1997. 288-301.
7. U. Çetintemel, P. J. Keleher, and M. J. Franklin. Support for speculative update propagation and mobility in Deno. In Proceedings of 21st International Conference on Distributed Computing Systems. Phoenix, Arizona: IEEE Computer Society Press, 2001. 509-516.
8. P. J. Keleher. Decentralized replicated object protocols. In Proceedings of 18th Annual ACM Symposium on Principles of Distributed Computing. Atlanta, Georgia, 1999. 143-151.
9. Huaizhong Lin and Chun Chen. Optimistic voting for managing replicated data. Journal of Computer Science and Technology, 2002, 17(6): 874-881.
10. M. M. Deris, A. Mamat, and M. P. Hamzah. Replicated data management for transactions sharing in distributed database. In Proceedings of 4th International Conference/Exhibition on High Performance Computing in Asia-Pacific Region. Beijing, China: IEEE Computer Society Press, 2000. 836-841.