システムソフトウェアと
オペレーティング・システム 71－17
マルチメディア通信と分散処理 73－17
（１９９５．１２．１）

Replica Control Protocol Based on Domain Leader Concept

Kenji Nakamura †, Yohtaro Miyanishi ‡, Fumiaki Sato † and Tadanori Mizuno †
† Shizuoka University
3-5-1 Johoku, Hamamatsu, 432, JAPAN
‡ Mitsubishi Electric Corporation,
7-10-4 Nishigotanda Shinagawa, Tokyo, 141, JAPAN

There are several algorithms of replica management in distributed database system, all it present the similar problem in trade-off of performance issue between read and write. Beside this problem, the available algorithms do not maintain the performance in system with great number of replicas. With the advance of technology, the utilization of replica in great scale tend to be important subject principally in system that involve multimedia application.

Our domain-leader algorithm have as objective to manage great quantity of replica maintained the reading performance similar to primary-copy and writing performance superior to quorum-consensus algorithm and promote flexibility to facilitate the concurrency execution of transaction in mobile system.

In addition, domain-leader improve the availability of system in occurrence of fail using the concept of priority and majority. In this paper, we present our domain leader algorithm, how the consistency is maintained between replicas, the performance comparison showing through simulation the domain leader and other two algorithm cited above, the future directions of research.

ドメインリーダコンセプトを用いた 複製管理プロトコル

中村 健二† 宮西 洋太郎‡ 佐藤 文明† 水野 忠則†
†静岡大学工学部 ‡三菱電機㈱

従来の複製管理プロトコルはリード/ライト性能のトレードオフ問題がある．また，複製数が多くなると性能が急激に低下する問題もある．現在，計算機技術の進歩により大量のデータを扱うマルチメディアアプリケーションを用いるシステムが続々と増え続けることから，多数の複製に対応できるプロトコルが必要になると思われる．

我々の提案したドメインリーダアルゴリズムは多数の複製を対称とし，性能改善，高度な可用性を保つことを目的とする．なお，本論文ではドメインリーダアルゴリズムの紹介，複製間での一貫性の保証，従来のアルゴリズムとの性能比較，故障発生時の対処，これからの研究課題を述べる．

I. Introduction

With the advance of technology, where every day the network is more and more rapid and the cost of resources is every time more cheap, as consequence the use of replica in great scale tend to improve.

Replica control protocol is used to hold the consistency of the data among the replicas and to control the access in the replicas in distributed databases. Transaction [10] are frequently effectuated in each site of the replicas, and to maintain the consistency of data, this transactions are propagated to other replicas.

There are several algorithms of replica management which the more known is primary-copy method [2] and quorum-consensus method [1, 3]. Moreover exist other algorithm as hybrid schemes this two[4, 5, 6, 7, 8].

In this paper we considerate the transaction as a primitive operation of reading and writing, where read mean retrieve of data; write mean operation that perform the retrieve of data, process and perform writing, i.e., operation of updating; the site is the node that contain the replicas of data.

II. Motivation and Goals

The two algorithm mentioned above, not satisfy completely in performance of reading and writing both at the same time. Other problem presented is that this algorithms has been developed within of limited network environment where trade-off of performance and cost was great problem, i.e., the network communication was slow and the resources was costly, the algorithm not foresee the system with great number of replicas. The flexbility fault

of this algorithm also is a problem at to be considered. If have a system which each site have several characteristic of reading and writing, the algorithm fit all system to support more reading or fit all system to wri-ting, compromising the general performance of system.

We algorithm objectify to resolve this problem introducing the concept of domain-leader which have next goal:

1) Ensure graceful degradation of performance in system with great number of replica;
2) Reduction of communication cost between replicas;
3) promote high availability in moment of fail.

Our research search solution and goals considering that each site have storage capacity, CPU and I/O; the cost of data storage is uniform to all sites; the network topology is ignored considering that network is all connected; all site contain a copy of data; the local databases design is ignored; only the data transmission is considered, i.e., locking and commit message and wait time is ignored; the local cost of processing is ignored.

III. Domain Leader Model

In system with great number of replica, is plenty complicated to maintain the consistency and correctness of data between replica without to pain the performance, principally in occurrence of fail.

Our domain-leader share the replicas in group which we call of domain (fig.1). Each site within of domain is attributed a numeric value of priority where site with major number is considered as leader. This assure that in situation of leader break, possibility other site with major priority to assume the place of leader as sub-leader.

The priority value is kept in a table called of *list_of_site* (fig.2a) which contain all sites within domain with respective priority value. This table is maintained in all sites where the content is different for each domain. Beside priority, is maintained in table a variable called *majority_of_site* which have the minimal number operable of site in domain. This assure in the moment of communication fail, where the sites is parted in partitions, at least one partition to be operable and only one site to be chosen as sub-leader.

All transaction executed by site is controlled by leader and only leader can communicate with other leaders. With this, domain-leader assure the concurrency execution of transaction within of domain and between other sites.

In domain-leader is used other table called of *list_of_leader* (fig.2b) which have the numeric value of priority of all leaders which leader with major priority is chosen as primary-leader. The primary-leader control the concurrency execution of transaction between leaders similarly as primary-copy method. The table *list_of_leader* is maintained in all sites of all domain unlike *list_of_site* which is alone to each domain. With use of priority, domain-leader allow in moment of break of primary-leader, other leader to be chosen as sub-primary-leader.

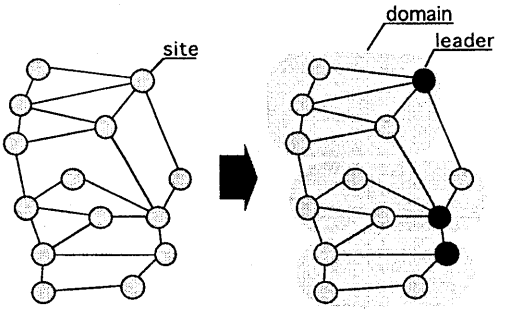

fig.1: Domain leader model.

As in *list_of_site*, the table *list_of_leader* contain a variable called of *majority_of_leader*. The concept of majority besides treatment of fail, also is used to improve the response time between leaders. That is, when primary-leader through multicast seek the approval of majority of leaders, not is needed the response of all sites.

Besides this tables, we attribute to all sites a variable called of *situation_of_site* which present two state, active and passive. This variable is used to avoid the overload in occurrence of fail, i.e., the site inaccessible by leader or by primary-leader, site is put in passive situation. The passive site is passed over by other sites, and only is returned to active state when site can communicate with leader.

Site(x)	priority
sx1	psx1
sx2	psx2
sx3	psx3
Sx	pSx
majority	mSx

sx: site that have data x
psx: priority of sx
mSx: majority of sx in domain

Leader(x)	priority
lx1	plx1
lx2	plx2
lx3	plx3
Lx	pLx
majority	mLx

lx: leader that have data x
plx: priority of lx
mLx: majority of lx in all replicas

fig.2: Table of list_of_site and list_of_leader.

In domain-leader we use to all communication between sites the 2PC (two phase commit protocol), 2PL (two phase lock) [9] to all operation of reading and writing. Finally, is used the file logT to maintain the consistency at moment of recovery after fail.

We explain in following with more detail how is proceeded the transaction when performed by client in one ofsites.

A. Behavior in site

When an operation of reading of the referenced data-item x is happened in site which not is leader, the transaction T is executed as showed below:

```
leader(x) = choose(list_of_site(site?(x)));
if site(x) not in list_of_site(x)  /*site(x) = current site accessed*/
  then access directly the leader;
  else begin site(x) = site;
            read(site(x), version(x));
            send(leader(x), site(x), version(x), T);
            version_of_site(x) = version(x);
            receive(leader(x), value(x), version(x), T);
            if version_of_site(x) = version(x)
              then read(site(x), value(x));
              else begin write(site(x), value(x), version(x));
                         write(site_logT, "actualized", T);
                   end;
       end;
```

fig.3: Behavior in site for read operation.

The site?(x) is a function that return a list contained all the sites of the referenced data-item x within of the domain. We frequently use the word within of the domain to remember that access not is made in all sites. Don't forget too that the leader is simply a site which have the higher priority compared with others. We use the word priority only to define the choice order of the leader and primary-leader.

The function choose(list_of_site(x)), choose the leader within of *list_of_site* to the data-item x accord with the priority. The choice also can vary accord with the condition of network, that is,the site or leader or communication can break for any fail. In this case, an other site with the higher priority but less than leader can be chosen as sub-leader. We explain afterwards with more detail the treatment during and after of fail.

If the site accessed not have the referenced data-item x, the access is made directly in the leader, if leader is locked, is waited until the unlock of leader. If the data x is founded in the proper site, is sent the version of data of site to the leader. When the version is devolved by leader is equal the version of local site, is read the value of x in local site. Else, the site is actualized with the new value sent by leader.

In following, is shown how is made the transaction of write request.

When write request is arrived in site, it send to the leader the message that want execute the writing. In receiving the response of the leader that can proceed, is sent the new value. Until receive the response of success or not of transaction, the value of data is written in a not volatile temporary disk [10] how order the 2PC. Note that the operation of writing is generally executed subsequently after of reading, but also can happen after a some time. In this case can be necessary the new search of leader of data-item x in list of site and after this, is sent the new value together with the version of data of transaction executed. Case transaction was executed over the version not actualized of data, the new value is read from leader and subsequently is made the reprocess and the writing.

Upon receive commit message from leader, is written definitively in site the new value of data item x together with the new version.

```
version_of_site(x) = version(x);
send(leader(x), "prepare", T);
receive(leader(x), message, T);
if message = "prepared"
  then begin send(leader(x), site(x), value(x), version(x), T);
             receive(leader(x), message, T);
             if message = "committed"
               then if site(x) in list_of_site(x) then
                      begin add 1 to version(x);
                            write(site(x), value(x), version(x));
                      end;
             write(site_logT, "committed", T);
  else write(site_logT, "aborted", T);
```

fig.4: Behavior in site for write operation

B. Behavior in leader

The leader is always involved in concurrency execution of events between sites and between leader to maintain the consistency and correctness of data. This events are performed how follow.

When the leader receive the read request of site, verify if the version of data of site is actualized or not activating the lock(x). If the version is equal, is immediately made the unlock(x) and a message is sent to site to indicate that data is actualized. Else is sent to site a new value of data-item x. Note that lock(x) and unlock(x) is a function which respectively lock and unlock the leader of referenced data-item x.

When the operation of reading is executed by proper leader, is verified if data is in lock or not. If is in lock, is waited until the unlock; else is read normally activating the lock and after the ending of reading is made the unlock. The request also can be sent by site that not have the data. In this case is treated as the request of proper leader.

Upon reception of write request performed by site, the leader will seek the primary-leader in leader?(x) through of choose(leader?(x)); leader?(x) is a function that return a list contained all of leaders of data-item x in priority order. Thereafter the primary-leader to be chosen, leader will send the message to primary-leader to indicate that is ready to write. At receive the message indicating that can proceed, leader will send the new value of data to the primary-leader and after response of commit, leader is

actualized. If the request was sent by site, then leader will send a message indicating the success of transaction to site. For facilitate the reading, we assume here that all of operation of reading and writing in the leader is made the lock, and unlock after the ending of transaction.

```
when receive read request from leader or other site which not
   have the data;
   read(leader(x), value(x), version(x), T);
   if site(x) != leader(x) then send(site(x), value(x), vers.(x), T);
when receive(site(x), version(x), T);
   version_of_site(x) = version(x);
   read(leader(x), version(x));
   if version_of_site(x) = version(x);
      then send(site(x), version(x), T);
      else begin read(leader(x), value(x));
                 send(site(x), value(x), version(x), T);
           end;
when receive(site(x), value(x), version(x), T);
   primary_leader(x) = choose(leader?(x));
   send(primary_leader(x), "prepare", T);
   receive(primary_leader(x), message, T);
   if message = "prepared" then
     begin send(primary_leader(x), value(x), T);
           receive(primary_leader(x), version(x), message, T);
           write(leader_logT, message, T);
           if message = "committed" then
             begin write(leader(x), value(x), version(x));
                   if site(x) not equal leader(x)
                     then send(site(x), "committed", T);
             end;
     end;
when receive(primary_leader(x), message, T);
   if message = "prepare"
     then send(primary_leader(x), "prepared", T);
when receive(primary_leader(x), value(x), version(x), T);
   send(primary_leader(x), "received", T);
   receive(primary_leader(x), message, T);
   if message = "committed"
     then begin write(leader(x), value(x), version(x));
                write(leader_logT, "committed", T);
          end;
```

fig.5: Behavior in leader.

When the request of writing received by leader has been sent by primary-leader, leader will send a response advising that is ready to receive the data. In receiving the new value of data, the leader is actualized and as soon as is sent to primary-leader the response of success of transaction.

C. Behavior in primary-leader

In primary-leader, the treatment of operation of reading is performed similarly how in leader. The writing is proceeded how showed down:

Upon receive the request of writing of leader, primary-leader will do the multicast of that want proceed the writing for the others leaders which is in list-of-leader of data-item x. Yonder leader, the request also can be of proper primary-leader or site which not have the referenced data x into domain. In this case, the primary-leader will do the multicast for all leaders.

```
when receive(leader(x), msg, T) or receive(site(x), msg, T);
  if msg = "prepare" then
    begin
      list_of_leader(x) = leader?(x);
      multicast(leader(x), list_of_leader(x), "prepare", T);
      leader_contacted(x) = receive(leader(x), message, T);
      if leader_contacted(x) >= leader_majority(x) then
        begin
          send(leader(x), "prepared", T);
          read(primary_leader(x), version(x));
          version_primary_leader(x) = version(x);
          when receive(leader(x), value(x), version(x), T);
            if version_primary_leader(x) not equal version(x)
              then the value received by leader is reprocessed;
            multicast(leader_contacted(x), value(x), T);
            leader_contacted(x) = receive(leader(x), msg, T);
            if leader_contacted(x) >= leader_majority(x) then
              begin
                send(leader(x), "committed", T);
                add 1 to version(x);
                write(primary_leader(x), value(x), vers.(x), T);
                write(primary_leader_logT, "committed", T);
              end;
  end;
```

fig.6: Behavior in primary-leader.

When the received response is majority of leaders, is sent the new value of data to the leaders. Upon receive the response of commit of majority of leader, the primary-leader will perform definitive save the data in it. If request was sent by leader, then commit message of transaction is sent to the leader.

Our domain-leader use concept of majority to hasten response time of writing, that is, the transaction of writing is performed as soon as have confirmation of majority of leaders. The consistency and correctness of data is maintained verifying the version which data was processed in the moment of arrival of write request of leader in primary-leader. If data was processed above of not actualized data caused by any fail, is sent the new version to leader and after reprocess, is made the writing.

D. Behavior during fail and recovery

We explain here how domain-leader manage the replica in moment of fail accordant the type or characteristic of fail through of behavior. We consider here that when table of *list_of_site* is actualized, the new value in table is sent to all sites communicable within domain. When *list_of_leader* table is actualized, the new value in table is sent to all leaders communicable. The function multicast

is not used only to verify if the group is formed by majority of leaders or sites, is also used to search the active leader or primary-leader, principally in moment of fail and recovery. We divide the type of fail in behavior which is showed below:

1. When the leader or primary-leader is broken or isolated for any fail, in moment of recovery:

a) the leader will do multicast to the other sites of domain indicating that is ready sending the version of your data. The message only is treated in sub-leader active. If the data of leader is not actualized, then the sub-leader will send the new value to leader. If have not response of active sub-leader, the leader will verify if can communicate with the majority of sites of domain.

b) If have response of majority of sites, then the leader will reassume your function sending to all leaders through of multicast that is active, together with version of your data. The message of leader is treated only by active primary-leader or sub-primary-leader. If the data of leader is not actualized, then will receive the new value from primary-leader.

c) Case adverse, the leader will communicate with primary-leader active through of multicast. If is possible the communication, then the leader assume that the sites of domain are isolate and perform the behavior 1.b. Else is conclude that the leader is isolated.

If the broken leader is fairly the primary-leader, repeat the step 1.a, 1.b, 1.c and 1.d.

d) The primary-leader will do the multicast to all leaders indicating that is active. If have the response of majority of leaders, the primary-leader will reassume your function actualizing the *list_of_leader* table.

2. When the site is broken or isolated, in moment of recovery:

a) the site will do multicast to the other sites of domain indicating that is ready sending the version of your data. The message only is treated in leader active. If the data of site is not actualized, then the leader will send the new value to site. If have not response of active leader, the site will verify if can communicate with the majority of sites of domain.

b) If have response of majority of sites, then the site will choose a site of major priority as sub-leader actualizing the table *list_of_site* and *list_of_leader*. The sub-leader will send to all leaders through of multicast that is active together with version of your data. The message of leader is treated only by active primary-leader or sub-primary-leader. If the data of leader is not actualized, then will receive the new value from primary-leader.

c) Case adverse, the site will communicate with primary-leader active through of multicast. If is possible the communication with active primary-leader, then the site assume that other sites of domain are isolated and perform the behavior 2.b. Else is conclude that the site is isolated.

3. When the leader do not achieve the communication with primary-leader:

If no achieve to communicate with the majority of leaders, the primary-leader wait until to get the communication with majority. In this case is made as behavior 1.

4. If leader do not achieve the communication with primary-leader:

The leader will do the multicast to other leaders, and if have response of majority, the leader will choose the major priority as sub-primary-leader and the table *list_of_leader* is actualized. Case have not the response of majority, the leader wait until to get the communication with majority. In this case is made as behavior 1.

5. If site do not achieve the communication with leader, then is executed as behavior 2.

IV. Comparison with other algorithms

We compare here, the communication cost of reading and writing between the our domain-leader with primary-leader and quorum-consensus method. In quorum-consensus, we use the dynamic adjustment of reading and writing quorum accordant the characteristic of system.

A. Communication cost

In primary-copy method, the processing of reading is done in proper site accessed which the cost is maintained zero, see fig.2. The writing is done for all sites saving the current site.

In quorum-consensus, the cost of reading and writing vary accordant the quorum definition. When the reading quorum is defined as 1, the cost of writing is considered S (total of sites or replicas), and when writing quorum is considered S, the cost of reading is 1. In resume, we can conclude that the medium of communication cost is maintained in S/2 to reading and writing.

The domain-leader maintain the medium 1 in cost of reading, due to access is done in proper site or in one leader. The writing is done in site accessed and all leaders where the cost is 1 + L(total of leaders). Note that the number of leaders is less than S/2.

As result, the cost of domain-leader is better than other two method in reading and writing.

method	Read	Write
primary-copy	0	S − 1
quorum-consensus (read quorum=write quorum)	S/2	S/2
domain-leader (2L<S)	1	L + 1

fig.7: communication cost between methods.

B. Simulation

We consider that the transaction events is arrived in each site in *poisson* order and the number of replica as 100. In domain-leader we consider the number of leader as 10 and for each leader we consider that exist 10 sites. We simulate in term of 200000 seconds where is done sampling at each 0.1 second time step. The comparison of result is showed in fig.8, fig.9 and fig.10. The fig.8 show the result of simulation when the number of read transactions arrived in site is very large. The fig.9 show the result when the transaction of write is superior than read transaction.

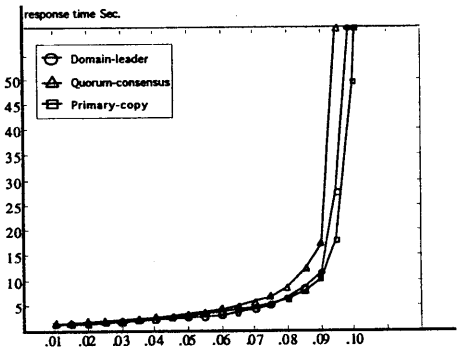

fig.8: read request response time when arrival probability is 90%.

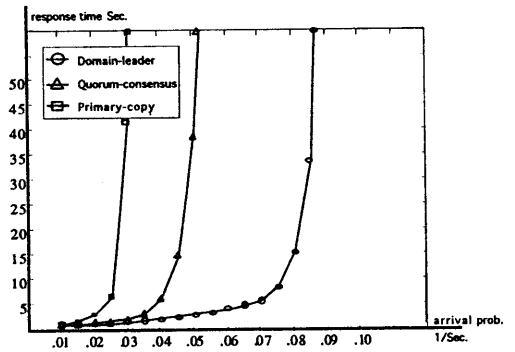

fig.9: write request response time when arrival probability is 80%.

In fig.10 is done the comparison using 10 replicas. Note that when the number of replicas is great, the difference of performance is more significant between domain and other two methods.

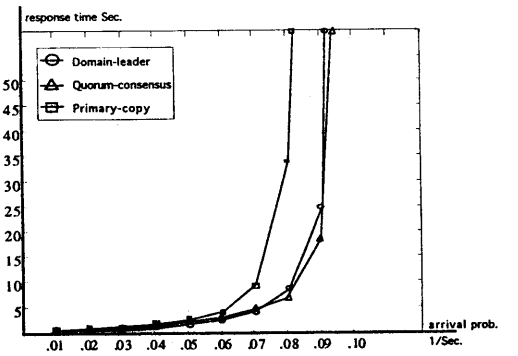

fig.10: write request response time when arrival probability is 80%(10 replicas).

V. Conclusion and Directions for Future Researches

Through method of division of distributed databases system in group which we called of domain, we achieves to improve plenty the performance of response time, principally in system that involve great quantity of replicas. Our domain-leader utilize two tables to maintain the consistency of data and to hasten more the response time of transaction execution. Our domain-leader show through simulation, good results compared to other algorithms. Principally in system that involve writing transaction.

Finally, due to our research is in initial phase, fault to verify with more detail the performance issue principally at time of system fail. The availability is a other topic that will be studied with more detail. Proof that the problem of bottleneck in leaders is despicable.

References

[1]D.Gifford, "Weighted voting for replicated data," *in Proc. 7th ACM Symp.Oper.Syst. Princip.*, Dec.1979, pp.150-162.

[2]B.M.Oki. and B.Liskov, "Viewstamped replication: A new primary copy method to support highly available dist. systems," *in Proc. 7th ACM Symp. Princip. Distrib. Comput.* Aug.1988, pp.8-17.

[3]M.Ahamad, M.H.Ammar, and S.Y.Cheung, "Multidimensional voting," *ACM Trans. Comput. Syst.*, Vol.9, No.4, pp.399-431, Nov.1991.

[4]M.Herlihy, "Dynanic Quorum Adjustment for Partitioned Data," *ACM Trans. Database Syst.*, Vol.12, No.2, June 1987, pp.170-194.

[5]A.El Abbabi and S.Toueg, "Maintaining Availability in Partitioned Replicated Databases," *Proc. Symp. Principles Database Systems (PODS)*, ACM Press, New York, N.Y., 1986, pp.340-351.

[6]S.Jajodia and D.Mutchler, "Dynamic Voting Algorithms for Maitaining the Consistency of a Replicated Database," *ACM Trans. Database Systems*, Vol.15, No.2, June 1990, pp.230-280.

[7]P.Triantafillou and D.J.Taylor, "The Location-Based Para-digm for Replication: Achieving Efficiency and Availability in Dist. Systems," *IEEE Trans. on Soft. Eng.*, Vol.21, No.1, Jan.1995, pp.1-18.

[8]A.Nakajima, "Decentralized Voting Protocols and their Communication Structures," *IEICE Trans. Inf. and Syst.*, Vol.E78-D, No.4, April 1995, pp.355-362.

[9]P.Bernstein, V.Hadzilacos, and N.Goodman, *Concurrency Control and Recovery in Database Systems*. Reading, MA: Addison-Wesley, 1987.

[10]B.Lampson, "Atomic transactions", *in Lecture notes in Computer Science vol. 105. Distributed Systems: Architec-ture and Implementation*. New York: Springer Verlag, pp.246-265, 1981.