view to the surviving backup HLRs, one that excludes the old primary HLR. The backup that replaces the primary HLR can be chosen by any function of that view. For example, the backup HLRs can choose the first member in that view as the replacement. That backup HLR can register itself as the primary HLR with a name service that the clients consult when they suspect that the primary HLR has failed (or when they require the service in the first place).

The second requirement is also satisfied, by the ordering property of view-synchronous semantics guarantee that either all the backup HLRs or none of them will deliver any given update before delivering the new view. Thus the new primary HLR and the surviving backup HLRs all agree on whether any particular client's update has or has not been processed. The primary-backup model may be used even where the primary replica HLR manager behaves in a non-deterministic way, for example due to multithreaded operation. Since the primary HLR communicates the updated state from the operations rather than a specification of the operations themselves, the backup HLRs slavishly record the state determined by the primary HLR's actions alone.

Passive replication has the disadvantage of providing relatively large overheads. View-synchronous communication requires several rounds of communication per multicast, and if the primary HLR fails then yet more latency is incurred while the group communication system agrees upon and delivers the new view. In variation of the model as presented here, VLRs may be able to submit read requests to the backup HLRs, thus off-loading work from the primary. The guarantee of linearisability is thereby lost but the BTSs receive a sequentially consistent service.

One of the main advantages of the primary-backup technique is to allow for non-deterministic operations [5]. The Sun Network Information Service (NIS, formerly Yellow Pages) uses passive replication to achieve high availability and good performance, although with weaker guarantees than sequential consistency. The weaker consistency guarantees are still satisfactory for many purposes, such as storing certain types of system administration records. The replicated data is updated at a master server and propagated from there to slave servers using one-to-one (rather than group) communication. Clients may communicate with either a master or a slave server to retrieve information. In INS, however, clients may not request updates: updates are made to the master's files.

2.2. Active Replication

Active replication, also called the state machine approach, is a non-centralized replication technique. Its key concept is that all replica HLRs receive and process the same sequence of VLR requests. Consistency is guaranteed by assuming that, when provided with the same input in the same order, replica HLRs will produce the same output. This assumption implies that server's process request in a deterministic way. VLRs do not contact one particular HLR, but address HLRs as a group. In order for HLRs to receive the same input in the same order, VLR requests can be propagated to HLRs using an Atomic Broadcast. Weaker communication primitives can also be used if semantic information about the operation is known (e.g., two request that commute do not have to be delivered at all HLRs in the same order).

The main advantage of active replication is its simplicity (e.g., same code everywhere) and failure transparency. Failures are fully hidden from the VLRs, since if a replica HLR fails; the requests are still processed by the other replica HLRs. The determinism constraint is the major drawback of this approach. Although one might also argue that having all the processing done on all replica HLRs consumes too many resources. Notice however, that the alternative that is, processing a request at only one replica HLR and transmitting the state changes to the others, in some cases may be much more complex and expensive than simply executing the invocation on all sites.

Figure 3 depicts the active replication technique using an Atomic Broadcast as communication primitive [4]. In active replication, Request phase and Server Coordination phase are merged and Agreement Coordination phase is not used. In the active replication for fault tolerance, the replica HLR managers are state machines that play equivalent roles and are organized as a group.

VLRs multicast their requests to the group of replica HLR managers and all the replica HLR managers process the request independently but identically and reply. If any replica HLR manager crashes, then this need have to impact upon the performance of this need have no impact upon the performance of the service, since the remaining replica HLR managers continue to respond in the normal way.

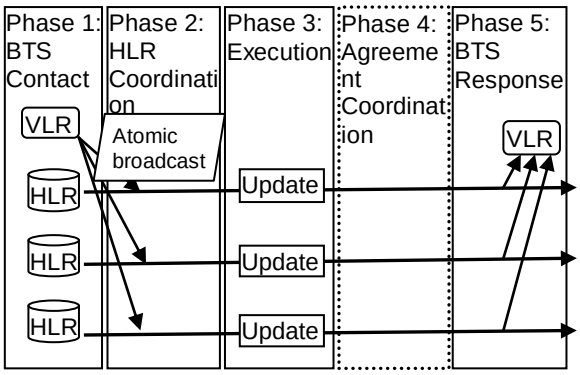

Figure 3: Active Replication

We shall see that active replication in tolerate byzantine failures; because the front end can collect and compare the replicas it receives.

Under active replication, the sequence of events when a VLR requests an operation to be performed is as follows:

- **Request:** The VLR attaches message to the request and multicast it to the group of replica HLR managers, using a totally ordered, reliable multicast primitives. The VLR is assumed to fail by crashing at worst. It does not issue the next request until it has received a response.
- **Coordination:** The group communication system delivers the request to every correct replica HLR manager in the same (total) order.
- **Execution:** Every replica HLR manager executes the request. Since they are state machines and since requests are delivered in the same total order, correct replica HLR managers all process the request identically. The response contains the VLR's unique request message.
- **Agreement:** No agreement phase is needed, because of the multicast delivery semantics.
- **Response:** Each replica HLR manager sends its response to the VLR. The number of replies that the VLR collects depends upon the failure assumptions and on the multicast algorithm. If, for example, the goal is to tolerate only crash failures and the multicast satisfies uniform agreement and ordering properties, then the VLR passes the first response to arrive back to the BTS and discards the rest (it can distinguish these from responses to other requests by examining the identifier in the response).

The system achieves sequential consistency. All correct replica HLR managers process the same sequence of requests. The reliability of the multicast ensures that every correct replica HLR management processes the same set of requests and the total order ensures that they process them in the same order. Since they are state machines, they all end up with the same state as one another after each request. Each VLR requests are served in FIFO order (because the VLR awaits a response before making the next request), which is the same as 'program order'. This ensures sequential consistency.

If BTSs do not communicate with other BTSs while waiting for responses to their requests, then their requests are processed in happened-before order. If BTSs are multithreaded and can communicate with one another while awaiting responses from the service, then to guarantee request processing in happened-before order we would have to replace the multicast with one that is both casually and totally ordered.

The active replication system does not achieve linearisability. This is because the total order in which the replica HLR managers process requests is not necessarily the same as the real-time order in which the BTSs made their requests. We have assumed a solution to totally ordered and reliable multicast. Solving reliable and totally ordered multicast is equivalent to solving consensus. Solving consensus in turn requires either that the system is synchronous or that a technique such as employing failure detectors is used in an asynchronous system, to work around the impossibility result of Fischer.

Some solutions to consensus, such as that Canetti and Rabin, work even with the assumption of byzantine failures. Given such a solution, and therefore a solution to totally ordered and reliable multicast, the active replication system can mask up to f byzantine failures, as long as the service incorporates at least 2f+1 replica managers. Each front end waits until it has collected f+1 identical response and passes that response back to the client. It discards other responses to the same request. To be strictly sure of which response is really associated with which request (given byzantine behavior), we require that the replica managers digitally sign their responses.

It may be possible to relax the system that we have described. First, we have assumed that all updates to the shared replicated HLRs must occur in the same order. However, in practice some operations may commute; that is, the effect of two operations performed in the order $o_1;o_2$ is the same as in the reverse order $o_2;o_1$.

For example, any two read-only operations (from different BTSs) commute; and any two operations that do not perform reads but update distinct objects commute. An active replication system may be able to explicit the knowledge of commutatively in order to avoid the expenses of ordering all the requests that some have proposed application-specific multicast ordering semantics.

Finally, VLRs may send read-only request only to individual replica HLR managers. In doing so, hey loss the fault-tolerance that comes with multicasting requests but the service remains sequentially consistent. Moreover, the front end can easily mask the failure of a replica HLR manager in this case, simply by submitting the read-only request to another replica HLR manager.

2.3 Semi-Active Replication

Semi-Active replication is an intermediate solution between active and passive replication. Semi-Active replication does not require that replica HLRs process service invocation in a deterministic manner. The protocol is originally proposed in a synchronous system model.

The main difference between semi-active replication and active replication is that each time replica HLRs has to make a non-deterministic decision, a process, called the leader, makes the choice and sends it to followers. Figure 4 depicts Semi-Active replication. Phase execution and agreement coordination are repeated for each non-deterministic choice.

The following steps characterize semi-active replication, according to our framework.

- **Request:** The VLR sends the request to the HLRs using an Atomic Broadcast.
- **Coordination:** The HLR coordinates using the order given by this Atomic Broadcast.
- **Execution:** All replica HLRs execute the request in the order they are delivered.
- **Agreement:** in case of a non-deterministic choice, the leader HLR informs the followers using the View Synchronous Broadcast.
- **Response:** The server HLRs send back the responses to the VLR.

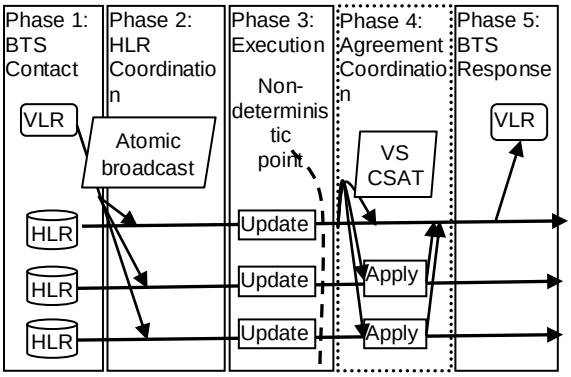

Figure 4: Semi-Active Replication

2.4. Semi-Passive Replication

Semi-passive replication is a variant of passive replication which can be implemented in the synchronous model without requiring any notation of views. The main advantage over passive replication is to allow foe aggressive time-outs values and suspecting crashed processes without incurring too high a cost for incorrect failure suspicious because this technique has no equivalence in the context of database replication. In semi-passive replication the Server Coordination (phase 2) and The Agreement Coordination (phase 4) are part of one single coordination protocol called Consensus with Differed Initial Values. Figure 5 summarizes the different replication approaches in distributed systems, grouped according two dimensions:

(1) Failure transparency for clients, and
(2) Server determination.

	Server Determinism Needed	Server Determinism Not Needed
Server Failure Not Transparent For the Clients		Passive
Server Failure Transparent For the Clients	Active	Semi-Active Semi-Passive

Figure 5: Replication in Distributed System

3. Conclusion

Except for location updating, most processing involves only a query operation in HLR. In other words, in most situations, the frequency of an HLR being queried is far greater than that of an HLR being updated. On the basis of this observation, we propose replicating the HLR to resolve the problem of HLR being the performance bottleneck and to increase the reliability of HLR. The most important issue in HLR replication is how many copies of an HLR must be replicated and where to place them. Replicating an HLR and distributing the replicas in the network can increase the reliability of HLR and reduce the communication cost as well as call setup time. On the other hand, replicating an HLR will increase the cost to setup and maintain database copies. Also, the database load will increase due to the extra overhead of update operations, making the contents of the databases consistent.

4. References

[1]. Guan-Chi Chen, Suh-Yin Lee, "Evaluation of Distributed and Replicated HLR for Location Management in PCS Network," Department of Computer Science and Information Engineering National Chiao Tung University Hsinchu, 300 Taiwan.

[2]. Y.B. Lin and I. Chlamtac, "Wireless and Mobile Network Architectures," Wiley, New York, 2001.

[3]. Jie Li, Yi Pan, Yang Xiao, "A Dynamic HLR Location Management Scheme for PCS Networks,".

[4]. M. Wiesmann, F. Pedone, A. Schiper, B. Kemme, G. Alonso, "Understanding Replication in Databases and Distributed Systems," Swiss Federal Institute of Technology (EPFL).

[5]. L.E. Mosar, P.M. Melliar-Smith, D.A. Agarwal, R.K. Budhiya, and C.A. Lingley-Papadopoulos, "A Fault Tolerance Multicast Group Communication", in April 1996.

[6]. Rachid Guerraoui and Andre Schiper, "Fault-Tolerance by Replication in Distributed System", in 1996.

[7]. Rachid Guerraoui, Andre Schiper, "Software-Based Replication for Fault Tolerance" Swiss Federal Institute of Technology, 1997.
[8]. Richard Golding, Elizabeth, "Fault-tolerance replication management in large-scale distributed storage systems".
[9]. Lantian Zheng, Stephen Chong, Andrew C. Myers and Steve Zdancewie, "Using Replication and Partitioning to Build Secure Distributed System".
[10]. Heinz Stockinger, Asad Samar, Bill Allcock, Ina Foster, Koen Holtman and Brain Tierney "File and Object Replication in Data Grids".
[11]. Ayalvadi J.Ganesh, Anne-Marie Kermarrec and Laurent Massoulie, "SCAMP: Peer-to-peer lightweight membership service for large-scale group communication".
[12]. Jeffrey O. Kephart, "Research challenges of Autonomic computing".
[13]. Jean-Charles Fabre and Tanguy Perennou, "A Metaobject Architecture for Fault-Tolerant Distributed Systems: The FRIENDS Approach".
[14]. Achmad I.Kistijantoro, Graham Morgan, Santosh K.Shivastava and Mark C.Little, "Component Replication in Distributed Systems: a case study using Enterprise Java Beans".
[15]. Paul D Ezhilchelvan, Raimundo A Macedo and Santosh K.Shrivastava, "Newtop: A Fault-Tolerant Group Communication Protocol".
[16]. Michael Steiner, Gene Tsudik and Michael Waidner, "Diffie-Hellman Key Distribution Extended to Group Communication".
[17]. Michel Cukier, Jennifer Ren, Chetan Sabnis, David Henke, Jessica Pistole, and William H.Sanders,"AQuA: An Adaptive Architecture that Provides Dependable Distributed Objects".
[18]. Mark C.Little, "Object replication in distributed systems".
[19]. Sandro Rafaeli and David Hutchison, "A Survey of Key Management for Secure Group Communication".
[20]. Bettina Kemme and Gustavo Alonso,"Don't be lazy, be consistent: Postgres-R, A new way to implement Database Replication".
[21]. Bruce Waliker, Gerald Popek, Robert English, Charles Kline and Greg Thiel,"The LOCUS Distributed Operating System".
[22]. Matthias Wiesmann, Fernando, Andre Schiper, Bettina Kemme and Gustavo Alonso,"Database Replication Techiniques: a Three Parameter Classification".
[23]. Franjo Plavec and Tomasz Czajkowski, "Distributed File Replication System based on FreePastry DHT".