Revisiting Hot Passive Replication*

Rubén de Juan-Marín, Hendrik Decker and Francesc D. Muñoz-Escoí
Instituto Tecnológico de Informática, Universidad Politécnica de Valencia
Camino de Vera s/n, 46022 Valencia, Spain
{rjuan, hendrik, fmunyoz}@iti.upv.es

Abstract

Passive replication has been extensively studied in the literature. However, there is no comprehensive study yet with regard to its degree of communication synchrony. Therefore, we propose a new, detailed classification of hot passive replication protocols, including a survey of the fault tolerance and performance of each class.

1 Introduction

Distributed replication is a means to increase the availability and dependability of computing systems. Active and passive replication can be distinguished [9]. For the passive (a.k.a. *primary-backup*) class [5], two main variants, *cold* and *warm* replication, are usually considered [24]. The *cold passive* approach does not require that any *secondary* or *backup* replicas were running processes. Instead, each request processed by the *primary* replica is checkpointed to secondary storage and all received messages are logged. In case of a primary failure, the logged information is recovered and used to restart a new primary. This ensures fault tolerance, but recovery typically takes long. The *warm passive* approach requires that all state updates generated by the primary are periodically propagated to the secondary replicas. Thus, the latter necessarily were running at failure time, so that recovery is faster. As a special case of the warm approach which merits further attention, *hot passive* replication propagates the state updates at the end of (and into) each update request. (Specifically for OLTP systems with two servers, another classification of primary-backup techniques is suggested in [14], which however does not easily generalize beyond its own framework.)

*This work has been partially supported by the Spanish grants TIC2003-09420-C02, TIN2006-14738-C02 and the EU project FP6-IST-004152 (DeDiSys).

The *cold* and *warm* variants have been carefully described and analyzed elsewhere (e.g., [10, 11, 12, 22]) and have also been considered in some standard specifications, for instance in FT-CORBA [12, 20, 22]. However, some initial formal studies of the passive model which analyzed the blocking time and the replication degree could not wholeheartedly recommend any of the two variants [5]. Thus, it comes as no surprise that other studies, though not related to CORBA, have only considered *hot* passive replication when comparing passive and active replication [1, 15]. Cold (and sometimes also warm) replication typically uses a rollback-recovery technique that necessitates a partial rollback of the client state in order to reissue the requests lost by a passively replicated server [10]. With *hot* replication, such problems are largely avoided.

Communication between servers may have a notable impact on the performance and behavior of hot passive replication. With that in mind, several classes of such systems can be distinguished as possible refinements of the "hot" approach. That precisely is the concern of this paper. Additionally, we also analyze the performance and failure handling of each identified subclass. Although active replication typically is the best solution when high availability is required, our results show that some refinements of the hot passive approach are not as slow as previously attributed to warm passive replication in general. Thus, we conclude that, in future, some of the hot passive subclasses identified in this paper could be considered when comparing or choosing between different replication alternatives of active or passive kind.

The rest of the paper is structured as follows. Section 2 outlines the system model. Section 3 defines passive replication. Section 4 addresses some communication issues as needed in the remainder. Section 5 identifies several subclasses of hot passive replication and analyzes their fault-tolerance, while section 6 studies their performance. Section 7 presents some related work. Section 8 concludes the paper.

2 System Model

We consider that all data are fully replicated in each network node. Interconnection channels are assumed reliable and with FIFO behavior (e.g., using TCP). In the following sections, different kinds of networks, varying in mean and extreme values of bandwidth and latency, will be considered. Moreover, we suppose that processes fail by crashing. We do not consider any kind of arbitrary (i.e., Byzantine) failures.

We assume the sequential consistency model [17]. In that model ..."*the result of any execution is the same as if the operations of all the processors were executed in some sequential order, and the operations of each individual processor appear in this sequence in the order specified by its program*". When such model is applied to a passively replicated process, all replicas should see the same sequence of updates, and this has also been assumed in the papers outlined in the previous section.

3 Principles of Hot Passive Replication

In replicated systems based on the hot passive approach, one server is selected as primary, while all others are backups. In this schema, clients send requests only to the primary, from where update propagation messages (*upm*, from now on) emanate, to all alive backups, once it has processed the request and before it sends an answer to the client. When the primary fails, one of the backups is promoted to the primary role. This approach abides by the following rules:

- *R1*. The primary role can only be owned by one replica at any time.
- *R2*. Only client requests received by the primary are processed.
- *R3*. The primary propagates updates to alive backups before answering the client request.

Additionally, a valid passive replication system [15] should ensure that no client request is lost in case of primary failure nor processed twice once a new primary replica is selected.

4 Communication

In the active replication model, a common requirement is that some *group communication system* (GCS, for short) needs to be used. To ensure consistency, that model needs total order delivery of requests sent by concurrent clients. Additionally, if easy recovery is asked for, the GCS needs to provide *virtual synchrony* [3, 7] (or *view-synchronous multicast* according to [15]) since it ensures that all replicas have delivered the same sequence of messages before any replica fails or any replica is added. According to [7], the most relaxed property related to multicast delivery that provides virtual synchrony is *same view delivery*; i.e., that all destinations of each multicast deliver each message when they belong to the same *view* (a view change arises when one process fails or rejoins the group).

At first sight, a GCS does not seem to be needed for passive replication. The most relaxed multicast primitive in a GCS is a reliable multicast ensuring that all alive destinations are able to deliver the multicast messages, which for passive replication should be complemented by FIFO delivery. Note that FIFO delivery is complex only in case of multiple senders, but that does not pose any problem for passive replication. Multicast is used for update propagation and is always initiated by a single (the primary) replica. In that case, FIFO is trivially ensured by message numbering. Moreover, reliable delivery can be ensured using retries even if channels are non-reliable.

But, on second thought, the question 'What with virtual synchrony?' arises. Should it be ensured? According to [15], it is indispensable for passive replication. Let us assume that no GCS is being used and that the requirements described in section 3 are respected, i.e., no client request is lost nor processed twice. Then, when the primary fails (a failure dectector [6] is assumed), the following steps should be executed:

1. A coordinator replica is chosen deterministically, e.g., the one with the lowest identifier.
2. Each alive replica sends the number of its last received *upm* to the coordinator.
3. The coordinator determines the maximum *upm* seen by any replica. If some replicas $R_1, ..., R_n$ have lost any *upm*, the coordinator requests from one of the replicas having all *upm*s (e.g., itself) to send the missed messages to the R_i.
4. Concurrently, all replicas have been able to consent on a new primary replica (e.g., the coordinator) and then, they send an acknowledgment message to that primary.
5. Once all acknowledgments have been received, the new primary is able to process new incoming requests.

The algorithm just sketched is functionally equivalent to the *flush* protocol described in [4] for ensuring

virtual synchrony, since it ensures that all messages multicast (and received by at least one replica) before the old primary crashed, will be delivered by all backup replicas before the new primary is able to start its service. Thus, a communication system (or a simple protocol) providing virtual synchrony is needed; the one outlined above is sufficient for that.

Note that, if the last *upm* sent by the crashed primary did not arrive to any backup replica, no problem arises. The client did not receive its associated reply and it will retry such request using the new primary.

In order to avoid that a request is processed twice (firstly by the failed primary and later by the new one, if the reply was lost), requests are numbered and their results can be included in the *upm*s and retained by the secondary replicas until needed. Some garbage collection technique should be added for discarding such retained results; some solutions to this problem have already been discussed elsewhere [19] and could be adapted to this setting.

Backup replicas should send an acknowledgment message (*ack*, for short) to the primary in order to ensure consistency among replicas and also determine the *blocking time* of a passive service, as stated in [5]. That paper characterizes *blocking time* as the *worst-case elapsed time between the receipt of a request and the sending of the associated response in a run that contains no failures*.

In order to study that blocking time, we should analyze when the *ack* messages are sent. Note, however, that in some systems, hot passive replication does not require any *ack* message, which results in a *non-blocking* (or *asynchronous*, or *lazy*) variant. Figure 1.a shows an example.

Some authors have analyzed different kinds of message-driven communication in terms of such *ack* messages, and defined *blocking* (or *synchronous* or *eager*) techniques. For instance, [23] distinguishes the following degrees of synchronicity and blocking:

- *reception*. In this case, a backup sends the *ack* as soon as it receives the *upm*.

- *delivery* (figure 1.b). Here, the *ack* is sent by each backup once it has delivered the *upm*. In our case, assuming that there is only one sender (the primary) and its message numbering procedure is able to ensure FIFO, there is no need to distinguish between reception and delivery. Other kinds of multicast primitives need such a differentiation, since additional message rounds are needed between message receipt and message delivery. So, in subsequent sections, only delivery is studied.

- *processing* (figure 1.c). This is the highest blocking level, in which the backup sends its *ack* only after it has processed the *upm*.

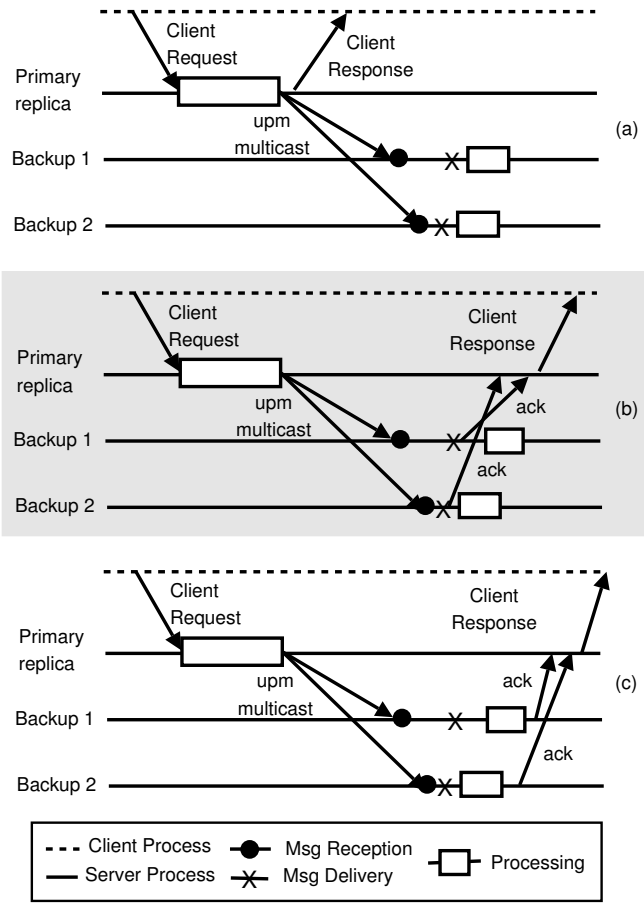

Figure 1. Blocking levels.

A possible optimization could be to block the primary until the first *ack* from a secondary arrives to it, instead of waiting for all the *ack*s. Then, according to the preceding discussion, *ack* messages might be sent on delivery or on processing, i.e., two new hot passive subclasses are obtained by this optimization. However, even with reliable channels, this optimization causes some problems. To begin with, if the network does not allow any low-level broadcast (e.g., in a WAN where IP-multicast is not possible), client requests may be lost. For instance, assume the primary has some internal synchronization problem and yet has sent a point-to-point request message to only a single secondary. Suppose the latter replies immediately and the primary gets its *ack* before it can send the next point-to-point *upm* to the next secondary. Upon receiving the *ack*, the primary replies to the client. If

then, both the primary and the updated secondary fail, the request is lost, because neither any new primary nor the rest of its backups will know anything about that request, since their states have not been updated accordingly. Even though this is very unlikely to occur, it might happen (think of Murphy's law) and should be prevented. The regular way to avoid it is to wait for all *ack*s.

So, this optimization based on a first answer is only possible if *uniform delivery* [16] is ensured. Uniform delivery means that if a multicast message has been delivered by any destination process (correct or not), all destination processes will be able to deliver that message. To ensure this, some extra message rounds are needed in order to guarantee the correct receipt of a given message prior to its delivery. This is extremely costly since it tends to require two more message rounds for all multicast messages, even when no failures arise. Figure 2 shows the behavior of this optimization.

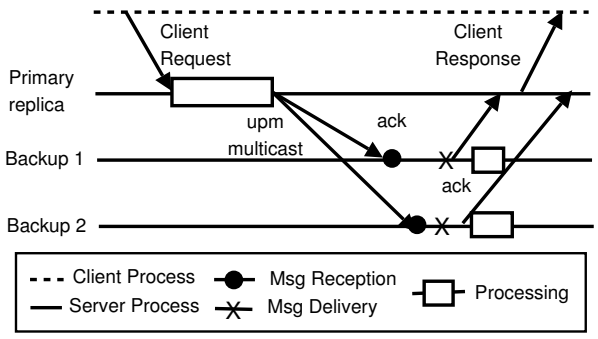

Figure 2. Blocking on first ack (on-delivery).

5 Classifying Hot Passive Replication

Table 1 shows a complete array of subclasses of hot passive replication, starting with the safest approach and terminating with the most relaxed one. With each of them, a higher degree of consistency than with warm passive replication can be achieved. But, as seen in section 4, not all of them can always ensure fault tolerance and fast recovery. Each subclass will be discussed subsequently, with special attention to their fault tolerance capabilities and their use in commercial or laboratory systems.

5.1 BP-AA

This subclass has been in the focus of many theoretical studies of the passive replication model [5, 15, 1]

Blocking on processing - all answers	BP-AA
Blocking on processing - first answer	BP-FA
Blocking on delivery - all answers	BD-AA
Blocking on delivery - first answer	BD-FA
Non-blocking	NB

Table 1. Hot passive subclasses.

and in some research systems (e.g., [26]). Here, the primary is not able to reply to its client until all secondary replicas have acknowledged the application of the *upm*s associated to that client request. Thus, this subclass entails the longest blocking time, while ensuring the consistency of all replicas. So, this is the safest option.

If the primary crash-fails, the client does not receive any reply and thus will take note of the crash sooner or later. Thus, it can resend its request to the new primary. If the previous primary was able to send any of its *upm* messages and a virtual synchronous multicast was used (as recommended above), all service replicas will share the same state, and the new primary can answer that request without needing to re-execute it. Recall that the *upm* messages should include a copy of the reply that the primary plans to send back to the client. On the other hand, if the previous primary did not send any *upm* message (e.g., it might have crashed before the local service of that request was terminated), none of the surviving replicas will know anything about such client request, so the new primary should execute it from scratch.

5.2 BP-FA

As already indicated in section 4, there is an optimization of the previous subclass that might produce better results in a WAN environment where the backup *ack*s and their preceding *upm* messages might take an undetermined, possibly long time. However, we have seen that this solution necessitates uniform *upm* delivery in order to guarantee that all backup replicas have seen the same sequence of *upm* messages and will finally have the same state. Without uniform *upm* delivery, this subclass is not able to overcome the simultaneous failure of primary and first answering backup nodes, since the delivery of the last *upm* by other backup replicas cannot be guaranteed. This situation has already been discussed in more detail in section 4.

In spite of this, a good solution for WAN environments is the "first-answer" approach of the three-tier replication scheme in [1]: positioned on a middle tier,

several tightly connected replicas enable a fast and bounded execution of all consensus protocols needed to guarantee message stability and uniform delivery. Once all replicas in the middle tier have consented, the request messages are propagated to the end-tier replicas, and the middle-tier replies to the client once the first end-tier answer is received. If part of the middle tier crashes during request processing, the remaining replicas are able to appropriately handle the rest of the request. For instance, if a failed replica has not terminated the multicast to the end-tier replicas, the remaining middle-tier replicas can take over and finish it.

The three-tier replication scheme does not exactly correspond to passive replication, but is easily adjustable to it. To do that, each replica node in a WAN can be implemented by a site composed of several independent nodes. If such a site represents the primary, its local nodes must share all decisions related to message multicast, but without sharing the passively replicated state. So, if the primary fails, its local companion nodes will be able to terminate the *upm* multicast to the backup replicas, ensuring thus its uniform delivery. Even if one of the backups fails, e.g., the one providing the first answer, the remaining ones would see the same sequence of *upm* messages. Such "middle-tier" companions (so called since they correspond to the middle tier of [1]), can for example be realized by any physically close computing device with sufficient memory.

The adjustment discussed in the previous paragraph becomes effective by taking the following steps: (1) The current primary should send to its "middle-tier" companions a copy of the *upm* to be multicast to the backup replicas. A uniform multicast should be used for this, the costs of which are negligible because the physically close companions are supposed to be connected by a LAN. (2) Once this local multicast is completed, it proceeds with the regular *upm* multicast to the backup replicas. (3) After that, it reliably sends a final message to its "middle-tier" companions in order to discard a last *upm* message.

Note that this last step could be executed concurrently with any other task. In case the primary crashes, one of its companions (now referred to as *comp-sender*) checks if there was any *upm* message that completed step (1) but did not arrive to step (3). If so, such message is multicast again by the *comp-sender*, and the backups should wait until such message is delivered. Since the primary has failed, and since the comp-sender is not one of its backups, a substitute backup must be appointed. If no *upm* multicast was interrupted by the primary crash, the *comp-sender* can simply multicast an empty message to all backups in order to allow them to continue.

Another argument in favor of the "first-answer" approach is the discussion of active replication in [15]. Only the first answer is needed for active replication, since client processes are assumed to use an atomic multicast for propagating the requests to the replicas. Atomic multicast ensures *atomicity* [15]: "*if replica J delivers m in view X, then every correct replica of X also delivers m*". Now, note that atomicity in this sense is equivalent to uniform delivery (cf. last paragraph of section 4). Thus, we effectively are requiring the same as [15] for a first-answer solution.

5.3 BD-AA

The second relaxation of the subclass presented in section 5.1 is to block only until the *upm* messages have been delivered. Indeed, this also guarantees that their associated backups will be able to apply such updates. The single event that might avoid such application would be the crash of the backups, but for completely avoiding the success of a given client request, all replicas would have to crash. If at least one backup is able to continue after all *ack*s have reached the primary, as required for BP-AA, that backup will be promoted to primary if all other replicas crash and will hold the same state as the old primary as soon as it has sent its last answer to its clients.

This relaxation provides an important performance advantage in systems with frequent, time-consuming update operations, such as replicated databases. Indeed, this is the usual protocol schema for all databases with a primary copy node as a master replica using *eager update propagation*, according to [13]. For instance, although the protocols described in [21] are significantly more complex than the BD-AA subclass, they coincide with the characteristics of BD-AA whenever transactions do not require multiple conflict classes, i.e., in many practical cases.

5.4 BD-FA

This fourth subclass needs the same node-to-site translation as suggested in section 5.2 for ensuring uniform multicast delivery, thus allowing that the primary replies as soon as it receives the first backup *ack*. Now, when this solution is applied to the *blocking on delivery* approach, instead of the *blocking on processing* approach, the result still complies with the hot passive model requirements since both BP-AA and BD-AA subclasses taken as the basis were correct.

5.5 NB

Protocols in the non-blocking subclass are not able to guarantee that any secondary replica has applied the most recent updates served by the primary. For instance, the primary might send its reply to the client and it might fail immediately, i.e., before sending the first *upm* to any backup replica. Thus, some client requests might get lost. Despite this, non-blocking is common for data replication in systems offered by commercial vendors as witnessed by many white papers (e.g., [18]), since they consider state consistency more important than losing some requests.

So, this approach shares some of the problems discussed in the introduction regarding *cold* and *warm* passive replication. Its performance (i.e., blocking time) is the best among all subclasses discussed in here, but the entailed problems are not affordable by many applications. Thus, we shall not consider this subclass in our performance study since it does not comply with the requirements stated in section 3.

6 Performance Study

We have compared the performance of the four acceptable subclasses identified in the previous section by simulation. More precisely, 50000 requests have been used to test each subclass, assuming a single-threaded primary replica and a variable number of backup replicas (ranging from 1 to 10, although normally only 1 or 2 are used [26] and it is hard to find more than 5). The primary service time has been modeled by an exponential distribution with a mean value of 25 ms.

Two kinds of updates have been considered and performed in the backup replicas: one with a mean value of 1 ms (short update) and another with a mean of 15 ms (long updates, as is typical in replicated objects with a lot of state, like databases). Both updates have also been modeled using an exponential distribution. Additionally, two network configurations have been used. The first one supposes a LAN environment whose latency has been modeled as a normal distribution with a mean of 0.5 ms and a deviation of 0.08 ms. The second one assumes a WAN environment with a mean latency of 5 ms and a deviation of 1ms (also modeled as a normal distribution).

As a result of this setup, four different configurations have been analyzed. The results are described in the sequel. Note that all subclasses based on the *first answer* principle have needed two additional message rounds to ensure uniform delivery. In both kinds of networks, such additional messages use a LAN-typical latency, according to the issues discussed in section 5.2.

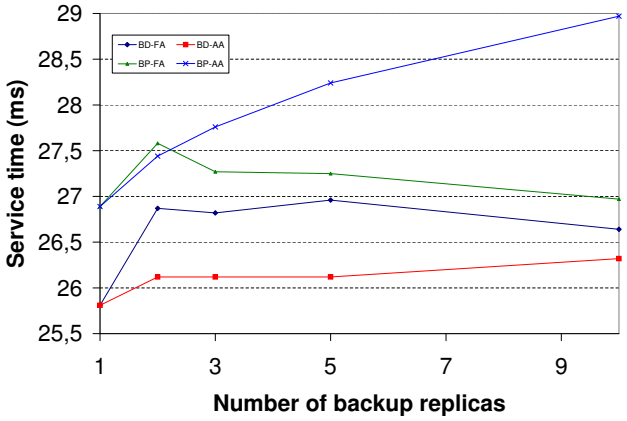

Figure 3. LAN results with short updates.

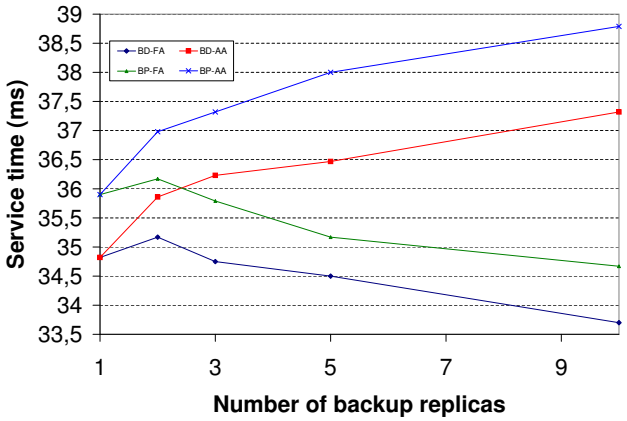

Figure 4. WAN results with short updates.

As expected, BP-AA yields the worst results in general, depending a lot on the performance of individual replicas. A single slow replica compromises the performance of BP-AA, and this leads to a big increase of the request service time when the number of backup replicas grows (see figures 3 through 6).

The same technique (BP-FA) proposed and measured in [1] for the active model in its three-tier replication architecture yields better results than BP-AA in our simulation. Additionally, it proportionally gets better the more backup replicas exist, and it does not depend on the configuration being tested, i.e., not on the network type nor on the length of the update operations applied in the backup replicas.

The second optimization proposed above (i.e., to block until delivery instead of blocking until processed) yields slightly better results than *blocking on*

processing, and this highly depends on the update application time consumption in the backup replicas, but also on the used network type. Using a LAN (see figures 3 and 5), the BD-FA provides higher service times than BD-AA, since BD-FA requires uniform delivery and this additional cost cannot be counterbalanced. And in such configurations, BD-FA and BD-AA yield results quite close to those of BP-FA.

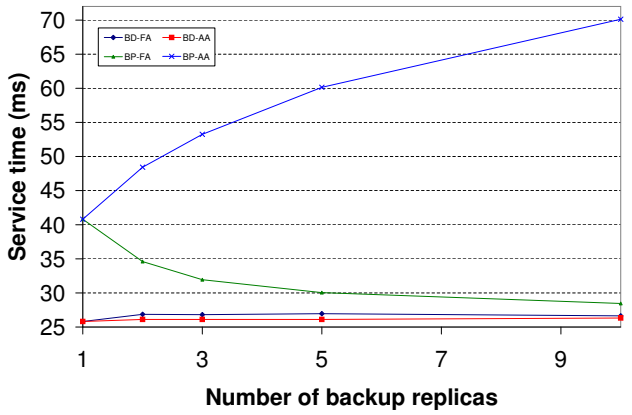

Figure 5. LAN results with long updates.

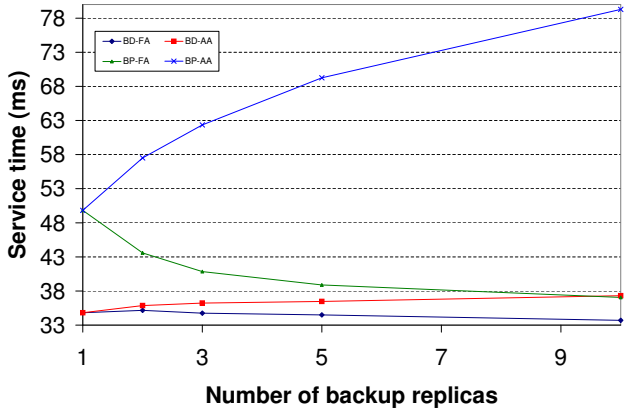

Figure 6. WAN results with long updates.

For WANs, the two BD-based subclasses feature similar performance trends as the BP-based ones, as shown in figures 4 and 6. Thus, BD-FA yields the best results, which get better when the number of backup replicas grows. On the other hand, since BD-AA scales worse than BP-FA, the BD-AA service time becomes longer than the one provided by BP-FA if many backup replicas are considered (3 in case of short updates in these examples, and 10 for long updates).

The performance results show that the three new hot passive subclasses discussed above might improve the typical hot passive performance in various distributed settings.

7 Related Work

Some related work has already been addressed above, since each of the discussed subclasses has either been considered already in other papers or can be easily derived from them. To our knowledge, no other work has jointly studied and compared the basic characteristics and the performance of all of these hot passive subclasses. Many, however, have distinguished and studied the cold, warm and hot classes of passive replication [11, 22, 24].

An issue not discussed in this paper that deserves further attention is recovery, i.e., algorithms to recover the state of a failed replica once it has been restarted. The regular solution consists in transferring the full state to such recovering replica, but this is not a smart solution when the replicated state is very big, as in a database. A recent paper [8] studies in detail the problems associated to the most generic transactional replication configuration: active replication with *linear interaction* [25]. Linear interaction implies that the results of each operation are transferred to backup replicas, as in the hot passive model. Our hot passive subclasses avoid the on-going transactions consistency problem as discussed in [8], because they have only one node that can process client requests at any time, even if interaction is linear. Other problems can be solved in our configurations as proposed in [8].

In the DeDiSys project [2], the use of the hot passive model has been generalized to partitionable environments. In such context, more than one primary may exist (indeed, one per partition), and this partially violates the requirements stated in section 3, but the associated problems have been solved by special reconciliation protocols. Thus, an extension of our hot passive classification is needed for properly dealing with partitionable systems, which will require some work. More precisely, a more relaxed definition of the passive model is needed for such scenarios.

8 Conclusion

We have revisited common classifications of the passive (or primary-backup) replication model, distinguishing cold and warm replication. Existing definitions of the passive model appear to restrict a lot the acceptability of passive replication variants, in particular those in the hot passive subclass. We have proposed two optimizations of some typical hot passive

replication deployments. The first one permits the backups to return control to the primary already upon delivery of the update message, instead of returning it only after completion of the update. The second one consists in waiting only for the first backup acknowledgment. We have simulated both approaches. The obtained results show that both optimizations improve typical deployments of hot passive replication in various distributed settings.

References

[1] R. Baldoni, C. Marchetti, and A. Virgillito. Impact of WAN channel behavior in end-to-end latency of replication protocols. In *6th European Dependable Computing Conference*, Coimbra, Portugal, Oct. 2006.

[2] S. Beyer, M.-C. Bañuls, P. Galdámez, J. Osrael, and F. D. Muñoz-Escoí. Increasing availability in a replicated partitionable distributed object system. In *ISPA*, pages 682–695, Dec. 2006.

[3] K. Birman and T. Joseph. Exploiting virtual synchrony in distributed systems. In *11th ACM Symposium on Operating Systems Principles*, pages 123–138, New York, NY, USA, 1987. ACM Press.

[4] K. P. Birman, A. Schiper, and P. Stephenson. Lightweigt causal and atomic group multicast. *ACM Trans. Comput. Syst.*, 9(3):272–314, 1991.

[5] N. Budhiraja, K. Marzullo, F. B. Schneider, and S. Toueg. The primary-backup approach. In S. J. Mullender, editor, *Distributed Systems*, chapter 8, pages 199–216. ACM Press, 2nd edition, 1993.

[6] T. D. Chandra and S. Toueg. Unreliable failure detectors for reliable distributed systems. *J. ACM*, 43(2):225–267, 1996.

[7] G. V. Chockler, I. Keidar, and R. Vitenberg. Group communication specifications: A comprehensive study. *ACM Computing Surveys*, 4(33):1–43, 2001.

[8] R. de Juan-Marín, L. Irún-Briz, and F. D. Muñoz-Escoí. Recovery strategies for linear replication. In *ISPA*, pages 710–723, Dec. 2006.

[9] X. Defago, A. Schiper, and N. Sergent. Semi-passive replication. In *Symposium on Reliable Distributed Systems*, pages 43–50, 1998.

[10] E. N. Elnozahy, L. Alvisi, Y.-M. Wang, and D. B. Johnson. A survey of rollback-recovery protocols in message-passing systems. *ACM Comput. Surv.*, 34(3):375–408, 2002.

[11] S. Garg, Y. Huang, C. M. R. Kintala, K. S. Trivedi, and S. Yajnik. Performance and reliability evaluation of passive replication schemes in application level fault tolerance. In *FTCS*, pages 322–329, 1999.

[12] A. S. Gokhale, B. Natarajan, D. C. Schmidt, and J. K. Cross. Towards real-time fault-tolerant CORBA middleware. *Cluster Computing*, 7(4):331–346, 2004.

[13] J. Gray, P. Helland, P. O'Neil, and D. Shasha. The dangers of replication and a solution. In *ACM SIGMOD Intnl. Conf. on Management of Data*, pages 173–182, 1996.

[14] J. Gray and A. Reuter. *Transaction Processing: Concepts and Techniques*. Morgan Kaufmann, San Mateo, CA, USA, 1993.

[15] R. Guerraoui and A. Schiper. Software-based replication for fault tolerance. *IEEE Computer*, 30(4):68–74, 1997.

[16] V. Hadzilacos and S. Toueg. Fault-tolerant broadcasts and related problems. In S. Mullender, editor, *Distributed Systems*, chapter 5, pages 97–145. ACM Press, 2nd edition, 1993.

[17] L. Lamport. How to make a multiprocessor computer that correctly executes multiprocess programs. *IEEE Trans. Computers*, 28(9):690–691, 1979.

[18] C. Mikkelsen and R. R. Schulman. Ensuring data integrity with asynchronous replication. HITACHI Data Systems white paper, July 2005. Downloadable: http://www.hds.com/pdf/wp_200_data_integrity_asynch_rep.pdf.

[19] F. D. Muñoz-Escoí, P. Galdámez, and J. M. Bernabéu-Aubán. ROI: An invocation mechanism for replicated objects. In *17th IEEE Symposium on Reliable Distributed Systems*, pages 29–35, West Lafayette, IN, USA, Oct. 1998.

[20] OMG. CORBA 3.0.3, Common Object Request Broker Architecture (Core Specification), 2004-03-01, Mar. 2004.

[21] M. Patiño-Martínez, R. Jiménez-Peris, B. Kemme, and G. Alonso. Scalable replication in database clusters. In *DISC*, pages 315–329, 2000.

[22] D. Szentiványi and S. Nadjm-Tehrani. Building and evaluating a fault-tolerant CORBA infrastructure. In *Workshop on Dependable Middleware-Based Systems (in DSN)*, 2002.

[23] A. S. Tanenbaum and M. van Steen. *Distributed Systems: Principles and Paradigms*. Prentice Hall, 2002.

[24] A. J. Wellings and A. Burns. Programming replicated systems in Ada 95. *The Computer Journal*, 39(5):361–373, 1996.

[25] M. Wiesmann, A. Schiper, F. Pedone, B. Kemme, and G. Alonso. Database replication techniques: A three parameter classification. In *19th IEEE Symposium on Reliable Distributed Systems*, pages 206–217, Oct. 2000.

[26] X. Zhang, D. Zagorodnov, M. A. Hiltunen, K. Marzullo, and R. D. Schlichting. Fault-tolerant grid services using primary-backup: feasibility and performance. In *CLUSTER*, pages 105–114, 2004.