Tunable Replica Consistency for Primary-Backup Replication in Distributed Soft Real-time and Embedded Systems

Jaiganesh Balasubramanian
*Zircon Computing
Wayne, NJ 07470, USA
Email: jai@zircomp.com

Aniruddha Gokhale
†Dept of EECS, Vanderbilt University
Nashville, TN 37235, USA
Email: a.gokhale@vanderbilt.edu

***Abstract*—In systems that use primary-backup replication for fault-tolerance, maintaining system availability after failures refers not just to ensuring the liveness of application functionality at a backup replica but also to ensuring that the state of the promoted backup matches that of the failed primary. Traditionally these availability criteria are realized in middleware through sophisticated algorithms that provide a certain level of replica state consistency, such as strong or weak. For distributed real-time and embedded (DRE) systems, the constant fluctuations in resource availabilities and application workloads, however, preclude a tight coupling to any single criteria for state consistency thereby forcing the need for new mechanisms that can tune the replica consistency algorithms at runtime in accordance with the operating conditions, and DRE system timeliness and availability requirements. This paper describes preliminary work in this space and highlights important challenges that must be addressed.**

keywords: real-time and availability, replica consistency, tunability.

I. Introduction

Distributed Real-time and Embedded (DRE) systems such as intelligence, surveillance and reconnaissance missions [1] consist of applications that participate in different end-to-end application flows. These systems operate in dynamic environments where new applications get introduced, existing applications terminate, processors and/or processes fail, and network and CPU resource availability fluctuates. Even when operating in such unpredictable environments, it is important to satisfy application quality-of-service (QoS) requirements, such as high availability and satisfactory response times of the soft real-time applications while simultaneously utilizing resources efficiently due to their limited number.

Although both ACTIVE and PASSIVE (*i.e.*, primary-backup) replication [2] are commonly available approaches for building fault-tolerant distributed applications that provide high availability and satisfactory response times for DRE systems, due to its low runtime overhead, passive replication is appealing for applications that cannot afford the cost of maintaining active replicas.

This work is supported in part by NSF award 0915976

The requirement to provide both high availability and satisfactory response times for clients in a passively replicated environment is conflicting in many ways. For example, to provide higher availability and strong state consistency, the state of a backup replica must be made consistent every time the state of the *primary* replica changes. This approach reduces failure recovery time since any one of the available *backup* replicas can be promoted as the new *primary* replica during failure recovery, and the clients can be be quickly redirected to the new *primary* replica.

However, this requirement adversely impacts response times perceived by client applications during the non-failure cases since the *primary* replica remains blocked until the state of all the *backup* replicas is made consistent with the state of the primary replica. In effect, the response times perceived by the client applications depend on the time taken by the primary to synchronize its state with that of the backup replica that operates in the slowest or most loaded physical host. Naturally, transient or delayed availability of network and CPU resources for state synchronization will further impact response times.

To overcome this limitation and to provide satisfactory response times for clients, a possibility is for the backup replica's state to be made consistent only intermittently or during failure recovery only, which significantly improves response times and saves resources, such as network bandwidth and CPU load, however, at the expense of a significantly weaker consistency model.

In summary, a range of alternatives are available ranging from strong to weak consistency to synchronize the state of the backup replicas with that of the primary. Each alternative can be characterized based on the response times provided to the clients, the recovery time after failures, the resources consumed, and the level of consistency provided by the application. Such a set of alternatives is acceptable to applications within the DRE system that have soft real-time requirements.

Although a number of prior efforts on middleware for both timeliness and availability exists, these research focus either on active replication-based systems [3] (which DRE systems cannot afford due to high resource overhead) or operate with a fixed strategy for state synchronization [4], [5] (which cannot deal with the dynamically fluctuating operating conditions). It is important therefore to be able to tune the consistency style of DRE systems at runtime in accordance with the operating conditions such that the combined requirements of system availability and client-perceived response times are at an acceptable level for the DRE system.

In the rest of this paper we propose the design and implementation of a middleware framework for analyzing and reasoning about tradeoffs that enable the tuning of replica consistency. We first present an overview of existing research in Section II. Following this, in Section III, we briefly present our approach alluding to the unresolved challenges that we are addressing. Finally we present concluding remarks in Section IV.

II. Related Research

Related research that focus on the tradeoffs between application performance, application fault-tolerance, and resource management are described below.

The real-time primary backup replication service [6] uses well-known scheduling algorithms, such as the rate monotonic scheduling algorithm [7], to schedule update tasks on the physical hosts deploying primary and backup replicas. Although powerful, a limitation of this approach is that the update schedule must be determined and fixed offline.

The scalable services architecture [8] organizes replicas in a chain. The head of the chain serves update requests, while the remaining replicas serve client read requests. All replicas except the head use gossip-based protocols to make their state consistent with the state of the replica at the head. However, all the replicas except the head, provide good performance by serving read requests rapidly for those clients that can tolerate weaker consistency.

AQuA [3] organizes the available replicas into two groups. The primary group is used for processing client update requests. The secondary group is used for processing client read requests. One of the replicas in the primary group *lazily* propagates its state to one or more of the replicas in the secondary group. Since different replicas in the secondary group might have different state, clients get weaker consistency assurances for their read requests. However, the read performance is good, as the replicas in the secondary group do not wait for the updates to be sequentially propagated from the primary group before processing client requests.

MEAD [9] employs a versatile dependability framework to change replication styles to vary the consistency assurances given for applications. For example, when more resources are available, active replication is used, and when resource availability is scarce, passive replication is used. Both MEAD and AQuA make runtime tradeoffs, however, state consistency mechanisms are not tunable.

IFLOW [10] uses passive replication with failure prediction techniques to determine the checkpointing frequency at which updates need to be propagated to the backup replicas. If the probability of the failure of the primary replica is high, the checkpoint frequency is also high. Under normal operating conditions, these optimizations reduce unnecessary checkpointing thereby improving application performance.

Our prior work on real-time fault-tolerant middleware also contains significant gaps. For example, *Fault-tolerant, Load-aware and Adaptive middlewaRe* (FLARe) [5] maintains service availability and soft real-time performance in dynamic environments. FLARe provides online failover in dynamic environments by changing the failover order of replicas according to the monitored utilization of resources and also provides overload management. However, FLARe assumes a fixed replica state update mechanism.

III. Tunable Adaptive Consistency for DRE Systems

To address the limitations of existing research in the field of optimizing resource usage for DRE systems while balancing response time and consistency, we are developing algorithms and mechanisms for a tunable and adaptive replica consistency management middleware. Our middleware schedules state update tasks depending on the available resources leading to different consistency levels for different applications.

A. System Model and Assumptions

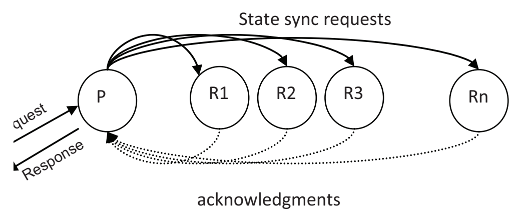

Figure 1. State Updates in Primary-Backup Replication

In this paper we assume that applications can specify the range of acceptable response times for their requests as part of their requirements. The goal of our adaptive fault-tolerant middleware is to execute the request in the primary replica, synchronize the modified state with the backup replicas, and send response to the clients within the acceptable response time bounds. We recognize that extra time is being spent in synchronizing the state of the primary replica with the backup replicas, and to satisfy the demands on response time, we can configure the number of replicas whose state is being synchronized.

To quantify this time, we propose to view state synchronization of the backup replicas from the primary replica as a sequence of end-to-end tasks [11], where the task chain starts at the primary replica, and ends at the last available backup replica as shown in Figure 1.

Each sub-task of the end-to-end task has a valid execution time, which is the time taken to set the state of the backup replica. The period of each sub-task is the same as the period of the main task which is the task that processes the client request.

B. Tunable Consistency Algorithm

Our tunable consistency algorithm is based on adjustments along two dimensions: (1) *consistency depth,* where we can adjust the number of replicas that will be made consistent with the primary on a state update request, and (2) *update frequency,* where the middleware decides how often the state of the primary should be made consistent with the backups.

We now explain the intuition behind our tunable algorithm. If the sum of the execution times of all the sub-tasks and the execution time of the client task at the primary replica is less than the response time deadline expected by the client, then our system can provide both high performance as well as high availability assurances for the applications. If the sum is more, then we reduce the length of the chain of the end-to-end tasks. This means that some of the backup replicas are not being synchronized with the state of the primary replica.

To determine how to bootstrap the system at deployment time, we apply the time-demand analysis to determine the maximum consistency level that can be provided for each of the applications given a deployment of primary and backup replicas on a set of physical hosts with some resource availabilities.

We realize that such a heuristic could try to provide maximum assurances for certain applications while starving some other applications. So we optimize this process by starting with a minimum end-to-end task length for all the applications. This means that initially all the applications are given a base consistency level. If all the applications are able to achieve that level, the consistency levels are slowly increased starting with the most important (*i.e.*, highest priority) application to the least important (*i.e.*, lowest priority) application in the entire DRE system.

C. Middleware Design

Our current work involves realizing the tunable replica consistency algorithm within a middleware that can work closely with the applications to determine when the state of the primary replica changes. Applications inform the consistency management middleware whenever the application's state changes. The consistency management middleware is armed with the consistency-related information computed from the time-demand analysis. Based on that information, the consistency management middleware knows how many backup replicas can be synchronized before the response needs to be sent back to the clients.

The consistency management middleware is integrated with the middleware replication manager of the FLARe middleware [5] so that it can obtain the references of the backup replicas and synchronizes their state. Finally, we also leverage the load utlization monitors provided in FLARe to determine how to adapt the consistency assurances provided to the applications depending on resource availabilities and workload fluctuations. This algorithm is shown in Figure 2.

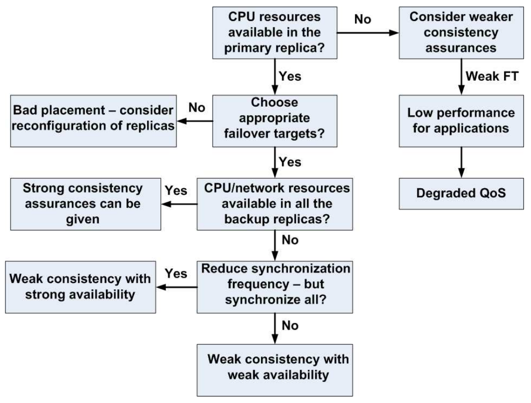
CPU resources available in the primary replica?
No
Consider weaker consistency assurances
Yes
Weak FT
Bad placement – consider reconfiguration of replicas
No
Choose appropriate failover targets?
Low performance for applications
Yes
Strong consistency assurances can be given
Yes
CPU/network resources available in all the backup replicas?
Degraded QoS
No
Weak consistency with strong availability
Yes
Reduce synchronization frequency – but synchronize all?
No
Weak consistency with weak availability

Figure 2. Runtime Adaptations of Replica Consistency

D. Unresolved Challenges

Many challenges remain unresolved when combining the strengths of our earlier work on FLARe with the new requirement of tunable consistency as explained below.

1. Conflicts between ranked list and synchronization order. Recall that FLARe defines a failover order of replicas based on the loads on the hosts. An interesting scenario may likely arise wherein the ranked list order computed by FLARe may not be the same order in which synchronization update requests may be feasible. For example, a ranked list for a primary P may be the tuple of replicas: $< A, B, C, D, E >$ in that order. However, it might be the case that it is not feasible to synchronize with B and C (because they cannot schedule their update tasks in a way that will meet response times) thereby leading us to a synchronization tuple of $< A, D, E >$, *i.e.*, a consistency depth of 3/5, which also has replicas in a order different from FLARe's ranked list.

Our proposed solution to address this challenge depends on obtaining additional hints from applications, through design-time tools such as model-driven tools.

2. Upper bound on consistency depth: Since consistency depth may vary over time, a natural question is what is the upper bound on the consistency depth our algorithm can provide to the client? Notice that our response time analysis is entirely driven by the dynamics of the system at that instant in time when a new task dynamically arrives in the system. We may find that 3 of the 5 replicas can be synchronized. Over time, due to changing loads in the system, we may be able to do at most 2 or even 1 replica. At other times we may be able to synchronize with 4 or even 5 replicas.

A proposed solution we are investigating involves defining design-time service level agreements (SLAs) with applications depending on their priorities. Naturally, these SLAs must be honored throughout the lifecycle of the application. Intuitively, if the rank list computed by FLARe is $< A, B, C, D, E >$ and a max consistency depth of 3/5 is agreed upon, then our middleware will ensure that state of the primary will be synchronized with anything from one replica $< A >$ to the tuple $< A, B, C >$ depending on existing loads.

3. Failures and/or significant changes to ranked list: Consider a 3/5 consistency depth for a ranked list of $< A, B, C, D, E >$. Thus, after a state update, A, B and C will have latest synchronized state. Now assume that due to changing dynamics, FLARe were to compute a new rank list that looks like $< D, E, C, A, B >$. At this point even if there is no failure, our depth of 3 will ensure that state is transmitted to $< D, E, C >$. Both D and E, however, had stale state since they were not in the preferred 3/5. If state is only incrementally updated as opposed to a full update, D and E are not the right choices to synchronize with since they cannot update their state.

Even if it is a full state update, things will go wrong on a failure. For example, consider a primary P dying and the ranked list changing to $< D, E, C >$ in the meantime. The first failover target $< D >$ does not have the most recent state and is useless to serve client requests (unless state can automatically be reconstructed).

A potential solution to resolve this dilemma is to let FLARe compute a ranked list that looks like: $< A, D, E, C, B >$. In other words, we modify FLARe's failover target selection from using the original "least loaded" principle to something that is a weighted mean of loads and the ability to schedule an update task. Despite a somewhat degraded performance by the newly chosen failover target, we rely on this weighted scheme since it provides more stronger forms of consistency. Note that we want to keep this strategy tunable on a per-application basis. Thus, the weights we use to compute the ranked list themselves can vary dynamically.

IV. Concluding Remarks

This paper described the need for tunable replica consistency as a necessary mechanism in middleware that provide both real-time and high availability to DRE systems. Many challenges remain to be resolved and our ongoing work is addressing these challenges. Although the tunable replica consistency approach builds upon our earlier work on FLARe (presented in RTAS 2009), the approach is general enough to be applicable to systems that employ passive replication schemes. In particular, we are exploring applying these solutions in our work on intelligent transportation systems.

References

[1] P. K. Sharma, J. P. Loyall, G. T. Heineman, R. E. Schantz, R. Shapiro, and G. Duzan, "Component-based dynamic qos adaptations in distributed real-time and embedded systems," in *CoopIS/DOA/ODBASE (2)*. Agia Napa, Cyprus: Springer, 2004, pp. 1208–1224.

[2] R. Guerraoui and A. Schiper, "Software-Based Replication for Fault Tolerance," *IEEE Computer*, vol. 30, no. 4, pp. 68–74, Apr. 1997.

[3] S. Krishnamurthy, W. H. Sanders, and M. Cukier, "An Adaptive Quality of Service Aware Middleware for Replicated Services," *IEEE Transactions on Parallel and Distributed Systems*, vol. 14, no. 11, pp. 1112–1125, 2003.

[4] R. van Renesse and F. B. Schneider, "Chain replication for supporting high throughput and availability," in *OSDI'04: Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation*. Berkeley, CA, USA: USENIX Association, 2004, pp. 7–7.

[5] J. Balasubramanian, S. Tambe, C. Lu, A. Gokhale, C. Gill, and D. C. Schmidt, "Adaptive Failover for Real-time Middleware with Passive Replication," in *Proceedings of the 15th Real-time and Embedded Applications Symposium (RTAS '09)*, San Francisco, CA, Apr. 2009, pp. 118–127.

[6] H. Zou and F. Jahanian, "A Real-time Primary-backup Replication Service," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 10, no. 6, pp. 533–548, 1999.

[7] C. Liu and J. Layland, "Scheduling Algorithms for Multiprogramming in a Hard-Real-time Environment," *JACM*, vol. 20, no. 1, pp. 46–61, Jan. 1973.

[8] T. Marian, K. Birman, and R. van Renesse, "A scalable services architecture," in *SRDS '06: Proceedings of the 25th IEEE Symposium on Reliable Distributed Systems (SRDS'06)*. Washington, DC, USA: IEEE Computer Society, 2006, pp. 289–300.

[9] P. Narasimhan, R. Rajkumar, G. Thaker, and P. Lardieri, "A versatile, proactive dependability approach to handling unanticipated events in distributed systems," in *IPDPS '05: Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05) - Workshop 2*. Washington, DC, USA: IEEE Computer Society, 2005, p. 136.1.

[10] Z. Cai, V. Kumar, B. F. Cooper, G. Eisenhauer, K. Schwan, and R. E. Strom, "Utility-Driven Proactive Management of Availability in Enterprise-Scale Information Flows." in *Proceedings of ACM/Usenix/IFIP Middleware*, 2006, pp. 382–403.

[11] X. Wang, C. Lu, and X. Koutsoukos, "Enhancing the Robustness of Distributed Real-Time Middleware via End-to-End Utilization Control," in *RTSS '05: Proceedings of the 26th IEEE International Real-Time Systems Symposium*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 189–199.