[42] P. Verissimo and J. Marques. Reliable broadcast for fault-tolerance on local computer networks. In *Ninth IEEE Symposium on Reliable Distributed Systems*, pages 54–63, Huntsville, AL, oct 1990.

[43] B. Walter. A robust and efficient protocol for checking the availability of remote sites. *Computer Networks*, 6(3):173–188, Jul 1982.

[28] H. Kopetz and W. Merker. The architecture of mars. In *15th Annual Symposium on Fault-Tolerant Computing*, pages 274–279, Ann Arbor, Michigan, Jun 1985.

[29] R. Ladin, B. Liskov, L. Shrira, and S. Ghemawat. Lazy replication: Exploiting the semantics of distributed services. Technical Report MIT/LCS/TR-484, MIT Laboratory for Computer Science, Cambridge, MA, Jul 1990.

[30] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, July 1978.

[31] B. Lampson. Designing a global name service. In *Proceedings of Fifth Symposium on the Principles of Distributed Computing*, pages 1–10, Aug. 1986.

[32] B. W. Lampson. Atomic transactions. In *Distributed Systems—Architecture and Implementation*, pages 246–265. Springer-Verlag, Berlin, 1981.

[33] R. Lemos and P. Ezhilchelvan. Agreement on the group membership in synchronous distributed systems. In *4th International Workshop on Distributed Algorithms*, Otranto, Italy, Sep 1990.

[34] S. Mishra, L. Peterson, and R. Schlichting. A membership protocol based on partial order. In *The Second IFIP Working Conference on Dependable Computing for Critical Applications*, pages 137–145, Tucson, AZ, Feb 1991.

[35] S. Mishra, L. L. Peterson, and R. D. Schlichting. Implementing fault-tolerant replicated objects using Psync. In *Eighth Symposium on Reliable Distributed Systems*, pages 42–52, Oct. 1989.

[36] B. M. Oki. Viewstamped replication for highly-available distributed systems. Technical Report TR MIT/LCS/TR-423, MIT Laboratory for Computer Science, Cambridge, MA, Cambridge, MA, Aug 1988.

[37] B. M. Oki and B. Liskov. Viewstamped replication: A new primary copy method to support highly-available distributed systems. In *Proceedings of the 7th ACM Symposium on Principles of Distributed Computing*, Aug 1988.

[38] L. L. Peterson, N. Buchholz, and R. D. Schlichting. Preserving and using context information in interprocess communication. *ACM Transactions on Computer Systems*, 7(3):217–246, Aug. 1989.

[39] D. Powell, D. Seaton, G. Bonn, P. Verissimo, and F. Waeselynk. The delta-4 approach to dependability in open distributed computing systems. In *Digest of Papers, The 18th International Symposium on Fault-Tolerant Computing*, Tokyo, Japan, Jun 1988.

[40] A. Ricciardi and K. Birman. Using process groups to implement failure detection in asynchronous environments. Technical Report TR 91-1188, Dept of Computer Science, Cornell University, Feb 1991.

[41] F. Schneider. Implementing fault-tolerant services using the state machine approach: A tutorial. *ACM Computing Surveys*, 22(4):299–319, Dec 1990.

[14] D. Davcev and W. A. Burkhard. Consistency and recovery control for replicated files. In *Proceedings of the Tenth ACM Symposium on Operating System Principles*, pages 86–96, Orcas Island, WA, Dec 1985.

[15] D. Dolev, J. Y. Halpern, and R. Strong. On the possibility and impossibility of achieving clock synchronization. In *Proceedings of 16th Anual ACM STOC*, pages 504–511, Washington D.C., Apr 1984.

[16] P. D. Ezhilchelvan and R. Lemos. A robust group membership algorithm for distributed real-time system. In *11th Real-Time Systems Symposium*, pages 173–179, Lake Buena Vista, Florida, Dec 1990.

[17] H. Garcia-Molina and A. Spauster. Ordered and reliable multicast communication. *ACM Transactions on Computer Systems*, 9(3):242–271, Aug 1991.

[18] D. Gifford. Weighted voting for replicated data. *Proceedings of the Seventh ACM Symposium on Operating System Principles*, pages 150–159, Dec. 1979.

[19] J. Grey. Notes on database operating systems. In *Lecture Notes in Computer Science 60*, pages 393–481. Springer-Verlag, Berlin, 1987.

[20] M. Herlihy. A quorum-consensus replication method for abstract data types. *ACM Transactions on Computer Systems*, 4(1):32–53, Feb 1986.

[21] M. Herlihy. Extending multiversion time-stamping protocols to exploit type information. *IEEE Computer*, C-36(4):443–448, Apr. 1987.

[22] N. C. Hutchinson and L. L. Peterson. The *x*-Kernel: An architecture for implementing network protocols. *IEEE Transactions on Software Engineering*, 17(1):64–76, Jan. 1991.

[23] D. Johnson and W. Zwaenopoel. Sender based message logging. In *Proceedings of the Seventeenth International Symposium on Fault-Tolerant Computing*, pages 14–19, Pittsburgh, PA, June 1987.

[24] D. Johnson and W. Zwaenopoel. Recovery in distributed systems using optimistic message logging and checkpointing. In *Proceedings of the Seventh Symposium on the Principles of Distributed Computing*, pages 171–181, Toronto, Canada, Aug. 1988.

[25] R. Koo and S. Toueg. Checkpointing and rolback-recovery for distributed systems. *IEEE Transactions on Software Engineering*, SE-13(1):23–31, Jan 1987.

[26] H. Kopetz, A. Damm, C. Koza, M. Mulazzani, W. Schwabl, C. Senft, and R. Zainlinger. Distributed fault-tolerant real-time systems: The Mars approach. *IEEE Micro*, pages 25–40, Feb 1989.

[27] H. Kopetz, G. Grunsteidl, and J. Reisinger. Fault-tolerant membership service in a synchronous distributed real-time system. In *International Working Conference on Dependable Computing for Critical Applications*, pages 167–174, Santa Barbara, California, Aug 1989.

consequence, reduces unnecessary interference among them; and (3) it allows the application to select exactly the right combination of building blocks to provide the required functionality. We believe that the importance of these advantages will increase as fault-tolerant systems become more prevelant in critical applications.

References

[1] P. A. Alsberg and J. D. Day. A principle for resilient sharing of distributed resources. In *Proceedings of 2nd International Conference on Software Engineering*, pages 627–644, Oct 1976.

[2] K. Birman. Replication and fault-tolerance in the ISIS system. In *Proceedings of the Tenth ACM Symposium on Operating System Principles*, pages 79–86, Orcas Island, WA, Dec. 1985.

[3] K. Birman and T. Joseph. Reliable communication in the presence of failures. *ACM Transactions on Computer Systems*, 5(1):47–76, Feb. 1987.

[4] K. Birman, T. Joseph, T. Raeuchle, and A. Abbadi. Implementing fault-tolerant distributed objects. *IEEE Transactions on Software Engineering*, SE-11(6):502–508, June 1985.

[5] K. Birman, A. Schiper, and P. Stephenson. Lightweight causal and atomic group multicast. *ACM Transactions on Computer Systems*, 9(3):272–314, Aug 1991.

[6] K. Birman, A. Schiper, and P. Stephenson. Lightweight causal and atomic group multicast. Technical Report 91-1192, Department of Computer Science, Cornell University, Feb 1991.

[7] A. D. Birrell, R. Levin, R. M. Needham, and M. D. Schroeder. Grapevine: an exercise in distributed computing. *Communications of the ACM*, 25 Nr 4:260–274, Apr. 1982.

[8] S. A. Bruso. A failure detection and notification protocol for distributed computing systems. In *Proceedings of the IEEE 5th International Conference on Distributed Computing Systems*, pages 116–123, Denver, CO, May 1985.

[9] J. Chang and N. Maxemchuk. Reliable broadcast protocols. *ACM Transactions on Computer Systems*, 2(3):251–273, Aug. 1984.

[10] F. Cristian. Agreeing on who is present and who is absent in a synchronous distributed system. In *Eighteenth International Conference on Fault-tolerant Computing*, pages 206–211, Tokyo, Jun 1988.

[11] F. Cristian. Understanding fault-tolerant distributed systems. *Communications of ACM*, 34(2):56–78, Feb 1991.

[12] F. Cristian, B. Dancey, and J. Dehn. Fault-tolerance in the advanced automation system. Technical Report Research Report RJ 7424, IBM Almaden Research Center, Apr 1990.

[13] D. Daniels and A. Z. Spector. An algorithm for replicated directories. In *Proceedings of the Second Annual ACM Symposium on Principles of Distributed Computing*, pages 104–113, Montreal, Canada, Dec. 1983.

7.3 Fault-tolerant Systems

We compare the system design of Consul with some of the recent fault-tolerant systems being developed. These include MARS [28, 26], AAS [12], DELTA-4 [39], and ISIS [6]. Both MARS and AAS are distributed real-time systems that employ synchronized clocks to implement various fault-tolerant services provided by the system. MARS is a system designed for distributed real-time process control applications, while AAS is designed to replace the present en-route and terminal approach U.S. air traffic control computer systems. Because these systems use synchronized clocks, the algorithms for various protocols in these systems cannot make use of the partial order among various events in the distributed system and hence they resort to more expensive total order. On the other hand, in Consul, partial order has been used to provide more efficient algorithms for these protocols, but no real time guarantees are made.

Isis and Delta-4 do not make use of the synchronized clocks for implementing various fault-tolerant protocols. The Delta-4 project seeks to define a dependable distributed, real-time operating system that allows integration of heterogeneous computing elements, while Isis is a distributed programming environment that provides tools for building fault-tolerant applications. Both of these systems provide causal ordering but they do not preserve the context graph and present it to the application. As a result, these systems cannot provide fault-tolerant algorithms that make use of the communication history of the system. In particular, weaker orderings such as semantic dependent ordering, that make use of the communication history, cannot be implemented in these systems.

A final advantage of Consul is that it provides a configurable architecture in which an application designer can build a system around a given collection of protocols with minimum effort. As a result, the system can satisfy the diverse needs of many different applications with little overhead and in a way that forces an application to pay for only the functionality that it needs. The use of reconfiguration protocols makes it easy to modify the system architecture or add new protocols to the substrate without affecting existing components.

8 Conclusions

This paper describes the design of Consul, a communication substrate for fault-tolerant, distributed programs. Consul consists of a suite of fault-tolerant communication protocols that together provide various fault-tolerant services such as broadcast, membership, and recovery. These protocols together form a substrate that can be used to build fault-tolerant applications. Specifically, the fault-tolerance support includes process failure detection, restart of failed processes, and reliable communication between processes. The support for distributed processing includes interprocess communication within a group of processes and different kinds of orderings among messages exchanged in the system.

This paper makes two major contributions. First, it gives novel algorithms for exploiting the partial ordering among the messages exchanged in the system. In particular, algorithms are given for semantic dependent ordering, membership, and recovery. Second, it suggests a decomposition of a fault-tolerant distributed system into a set of primitive building blocks. Decomposing a system into its fundamental building blocks has three advantages: (1) it simplifies the process of implementing, debugging, and optimizing each piece; (2) it aids in finding the inherent dependencies among the pieces, and as a they involve a collection of different kinds of operations, some requiring total ordering and some requiring partial ordering with respect to one another. Our approach performs much better in a case such as this when mixture of these different operations needs to be applied to an object. The approach proposed in [29] must resort to a total ordering in such a case.

7.2 Membership

Membership protocols have been proposed for both synchronous and asynchronous environment. Membership protocols in synchronous systems include [10, 16, 27, 33, 43]. All these protocols make use of synchronized clocks to maintain a consistent view of which processes are functioning at every clock tick.

Because the concepts of asynchrony and membership are incompatible, the membership problem is more difficult in asynchronous systems than in synchronous systems. The protocols proposed in [3 , 8, 9, 34, 40] and the one proposed in this paper assume an asynchronous environment. The protocol proposed in [8] communicates failure information by diffusion. This algorithm, however, does not attempt to maintain a consensus view of the configuration. The protocols proposed in [3, 9, 40] do maintain a consistent view. However, in all of these approaches, the complete protocol has to be restarted when a process fails while the protocol is in progress. On the other hand, our protocol manages such failures differently—failures or recoveries detected while the protocol is in progress are taken into account incrementally by updating **SuspectUpList** or **SuspectDownList** appropriately. Moreover, the protocol proposed in [40] only establishes a consistent time when a failed process is to be removed. In particular, it assumes that detecting and establishing the failure of a process is implemented elsewhere. Since in asynchronous systems it is impossible to distinguish with certainty between a failed processor and one that is merely slow, the best that can be done is reaching a tentative conclusion about a process that is suspected to have failed. Such a conclusion is reached by using some heuristics that typically involve communication among all the processes, for example, by using ack and nack messages, as we do above.

Another advantage of our protocol relative to the other approaches is that it relaxes the requirement that removal of a failed process from the membership list be totally ordered with respect to all other events. In particular, a process waits to update its membership list only until it has determined the last message sent by the failed process; it need not wait for other processes to update their membership lists. In contrast, other protocols force a process to wait until all functioning processes have confirmed the failure.

A final advantage is that removal of failed processes from the membership list need not be done at the same time at all the processes. This results from the fact that sf-groups are created dynamically at each process, and these groups need not be the same at all processes. Thus, a process that does not have to merge two sf-groups will be able to remove the members of the first group before another process that does have to merge the two groups. This improves the efficiency of the application and simplifies the design of the protocol. In contrast, other protocols wait until all the processes have formed their sf-groups before removing the failed process.

for this is that the system performs fewer operations per unit time when the rate of issue of operations is lower, leading to more idle time to do the checkpointing. As a result, its effect on the time to process an operation is less.

The second observation is that the overhead of checkpointing increases as the checkpoint interval is reduced. Thus, this overhead is 0.1 sec per 100 operations when operations are issued at 50 ops/sec and the checkpointing is done every 5 sec, while this overhead increases to 0.35 sec per 100 operations when the logging is done every 1 sec. This is expected, as the time spent to do the checkpointing per unit time increases as the checkpoint interval is reduced. The effect of checkpointing is almost negligible for checkpoint intervals of 5.0 sec or higher for the observed operation rates.

7 Related Work

Considerable attention has been given to the design of various fault-tolerant protocols and systems. In this section, we compare our work with some of the recent work being done in this area. As explained before, we have presented novel algorithms for semantic dependent ordering and membership in Consul, and therefore, in the following we compare these two functions to the related approaches in the literature. Finally, we compare the overall system design of Consul with some other fault-tolerant systems that also provide these fault-tolerant services.

7.1 Message Ordering

Our approach in managing replicated objects is similar in many respects to approaches taken elsewhere. These approaches may be classified into two categories. The first category includes those protocols where the semantics of the operations are not exploited and a total order is imposed to implement replicated objects or a related constructs. Examples of this approach include [1, 4, 2, 7, 18, 20, 31, 37, 36]

In the second category, the semantics of the application have been exploited to come up with a solution. In [13], semantic information has been used to implement a replicated directory, while in [14], the authors use semantic information to implement replicated files. Our approach differs from these two in that we maximize the concurrency by dividing different operations into op-groups, and our approach generalizes easily beyond files and directories. In [21], semantic information is used to efficiently implement multiversion timestamping protocol for atomic transactions. While this work is similar to ours, the two approaches differ in two aspects. First, we efficiently implement operations on an object instead of atomic transactions. Second, we deal with objects replicated over multiple sites. That is, our emphasis is on increasing concurrency of independent operations over multiple sites rather than increasing concurrency among transactions on a single site.

Finally, we compare our work with [29]. Here, lazy replication has been proposed as a way to preserve consistency by exploiting the semantics of the service's operations to relax the constraints on ordering. Three kinds of operations are supported: operations for which the clients define the required order dynamically during the execution, operations for which the service defines the order, and operations that must be globally ordered with respect to both client-ordered and service-ordered operations. While this approach is best suited for client-defined ordering, many applications do not fit in this model. Rather, protocol demultiplexes every message to the Failure Detection protocol and the Order protocol.

6.4 Checkpointing Overhead

An experiment to measure the checkpointing overhead is done as follows. Two clients at different processors issue operations at a known, fixed rate, and the elapsed time for every 100 operations is measured. In the experiment, 5,000 operations were issued by each client. Note that the time measured in this experiment cannot be used to calculate the response time since it is dominated by the time taken to issue the operations. However, since the operations are issued at a fixed rate, the effect of the checkpointing overhead is uniform over all the operations. This experiment includes both the asynchronous message logging done by Psync and the checkpointing done by the order protocol. For the former, a given message's identifier, dependencies, and contents are stored. For the latter, the message identifiers of all the messages in the participant's view are checkpointed; this is done atomically when a wave becomes complete.

The time measured is for four different rates of operations issued by the clients. For each rate, the elapsed time for 100 operations is measured under different checkpointing intervals. The results are shown in Table 4. All the operations issued were commutative operations, with the semantic dependent ordering protocol being used throughout.

Ops/sec	**Checkpoint interval (in sec)**	**Time for 100 Ops (in sec)**
10	No Checkpointing	10.0
10	5.0	10.0
10	2.0	10.0
10	1.0	10.01
20	No Checkpointing	5.99
20	5.0	6.0
20	2.0	6.035
20	1.0	6.07
40	No Checkpointing	4.0
40	5.0	4.05
40	2.0	4.15
40	1.0	4.25
50	No Checkpointing	2.0
50	5.0	2.1
50	2.0	2.3
50	1.0	2.35

Table 4: Measure of Checkpointing Overheads

Two observations can be made from these measurements. First, the checkpointing overhead increases with the increase in the rate at which clients issue operations. For example, the overhead is 0.35 sec per 100 operations when the clients issue operations at 50 ops/sec and the checkpointing interval is 1 sec, while the overhead is 0.01 sec per 100 operations when operations are issued at 10 ops/sec. The reason protocol when all the operations are noncommutative. That is, the overhead of the protocol is negligible, leading to minimal effect on system performance. Similar improvement was observed for the 3-replica and 4-replica systems. These results are shown in Table 2.

% of comm. operations	**Semantic Dep. Order**	**Total Order**
0	3.7	3.6
50	3.55	3.6
75	3.2	3.6
90	2.9	3.6
99	2.7	3.6
100	2.7	3.6

Table 1: System Response Time (in msec) for a 2-replica system

% of comm.	**Semantic Dep. Order**		**Total Order**	
operations	**3-replica**	**4-replica**	**3-replica**	**4-replica**
0	4.1	4.45	4.0	4.3
100	2.75	2.8	4.0	4.3

Table 2: System Response Time (in msec) for a 3-replica and 4-replica system

6.3 Membership and Recovery Protocols

The overhead of the failure handling protocols in the absence of failures is measured by extending the configuration to include the Membership, Failure Detection, and Recovery protocols. The code size for these protocols is only about 1500 lines, mainly because of the inherent support Psync provides for such activity by maintaining a replicated context graph. Once again, none of the protocols do any logging or checkpointing in this experiment. The response time was measured in the same way as described above for various mixes of commutative and noncommutative operations. The results are shown in Table 3.

The overhead imposed by the failure handling protocols is about 0.6 msec per operation. This overhead is due to two factors. First, since the communication substrate includes more protocols, the Divider protocol has a larger set of protocols to which to demultiplex incoming messages. Second, the Failure Detection protocol needs to receive every message exchanged in the system. Thus, the Divider

% of comm. operations	**Response Time**
0	4.2
50	4.1
75	3.8
90	3.6
99	3.3
100	3.2

Table 3: Response Time with Failure Handling Protocols (in mesc)

for two, three and four replicas. These configurations differ in several ways. One is the type of ordering protocol used; some use semantic dependent ordering, while others use total ordering. Another is whether or not they contain various failure handling protocols. A third is whether checkpointing is performed and at what interval. Our experience has been that it is easy to move from one configuration to another without any modifications to the substrate.

This section reports on the performance of various protocols in Consul and the overheads they impose on the overall performance of the system. All of the numbers reported here have been taken from the replicated directory application running on a collection of diskless Sun 3/75 workstations connected by a lightly loaded 10Mbs Ethernet; stable storage is simulated by a process executing on another Sun 3/75. Various experiments were designed to measure the performance of Psync and the semantic dependent ordering protocol, as well as the overhead of the various failure handling protocols.

6.1 Psync Timings

To measure the performance of Psync, one byte messages were exchanged between a pair of user processes directly on top of Psync. In this test, the resulting average round trip delay was measured as 2.9 msec. This number is derived by exchanging messages for 10,000 trips (20,000 total messages) and reporting the elapsed time for every 1,000 round trips. Each of these measurements was then divided by 1,000 to produce the average.

6.2 Performance Using Semantic Dependent Ordering

To determine how well the semantic dependent ordering protocol performs, we compared the performance of the replicated directory using the semantic dependent protocol with the same application using a total ordering protocol. In this experiment, we focused on measuring the average *response time* of the system, i.e., the elapsed time between the time an operation is issued by the client and the time that operation is applied to the local copy of the directory. The time needed to actually perform the operation is not included.

For this experiment, the communication substrate was configured to include Psync, the Divider protocol, and the appropriate order protocol. There was no logging or checkpointing done by any of these protocols. The system was configured to run on two, three and four processors respectively. In the case of the semantic dependent ordering protocol, the average response time depends heavily on the overall mix of the commutative and the noncommutative operations, so the mix was varied across different runs. In each case, the response time is derived by having clients on each processor apply 10,000 operations, with a varying percentage of commutative operations uniformly distributed, and reporting the elapsed time for every 1,000 operations. Each of these measurements was then divided by 1,000 to produce the average response time.

The results for the system configured for two replicas are shown in Table 1. As expected, the semantic dependent ordering protocol improves the response time of the system as the percentage of commutative operations increase. The response time is 2.7 msec when all the operations applied are commutative, giving an improvement of about 25% over the use of a total ordering protocol. Another important point to note is that the semantic dependent ordering protocol performs almost as well as the total ordering