Tracking Service Availability in Long Running Business Activities

Werner Vogels

Dept. of Computer Science, Cornell University, Upson Hall, Ithaca, NY 14853
vogels@cs.cornell.edu

Abstract. An important factor in the successful deployment of federated web services-based business activities will be the ability to guarantee reliable distributed operation and execution under scalable conditions. For example advanced failure management is essential for any reliable distributed operation but especially for the target areas of web service architectures, where the activities can be constructed out of services located at different enterprises, and are accessed over heterogeneous networks topologies. In this paper we describe the first technologies and implementations coming out of the Obduro project, which has as a goal to apply the results of scalability and reliability research to global scalable service oriented architectures. We present technology developed for failure and availability tracking of processes involved in long running business activities within a web services coordination framework. The Service Tracker, Coordination Service and related development toolkits are available for public usage.

1 Introduction

One of the driving forces behind the emerge of service oriented computing is the desire to use services architectures to construct complex, federated activities. When composing a business activity out of services that are heavily distributed a strong need arises to provide guarantees about the reliability and availability of the components that make up the activity. In the same context one wants to ensure that the aggregated service can handle failures of the individual services gracefully. To provide these advanced guarantees, the web service architectures can draw from the experiences of two decades of advanced middleware systems development, which had similar reliability goals, albeit at a smaller scale and in less heterogeneous settings.

Some first steps in providing an interoperable framework for distributed message oriented services have been set with the security, routing, coordination and transaction specifications [2,3]. Together with the reliable messaging specifications these will provide the first building blocks for constructing simple forms of federated web services based activities. Although these specifications do a reasonable job in laying the ground work for the interaction between the various services and the guarantees that can be achieved by making all of the distributed operations explicit, they are remarkably vague with respect to failure scenarios and failure handling.

The issue of failure management becomes especially important when we are considering long running business activities in federated setting. These activities are often the aggregate of several distributed services, and some may have running times of several hours. For this class of systems it becomes essential to provide the activity composer with the right tools to track the availability of the individual services and to provide mechanisms for service selection to support various styles of compensating actions in the case of failures. Long running activities need to deal with more complex failure scenarios than those that can be implemented using simple atomic transactions. For example a failure of a crucial service may need to be compensated by trying to use an alternative supplier or a replica service, before deciding on the rollback of the overall activity.

In the context of the *obduro* project we are developing a range of support tools for managing complex federated service execution. One of basic tools necessary to construct more complex activities is the Service Tracker, which is used to address the need for failure management of especially longer running activities.

The distributed systems core of the Service Tracking system has two fundamental components: failure detection and membership information dissemination. A variety of algorithms for both components has been developed over the years, but mainly for research systems, while industrial systems in general have resorted to simple heartbeat, time-out and flooding schemes [4,6,8,10,13, 14,15]. The prototype of the service tracking engine developed for this project is extremely well suited for cases where robustness and scalability are important. The prototype is based on epidemic techniques, and the a-synchronous and autonomous properties of this system make it simple to implement and operate.

Failure detection can also be used as the building block to simplify the implementation of other essential distributed systems services such as *consensus* [5]. Consensus is used when a set of processes need to agree upon the outcome of an operation. Many consensus protocols require knowledge about which processes are involved in the execution of the protocol to establish a notion of majority, quorums, etc. Even though some protocols such as Paxos [11,12] do not use failure detectors in their specification, the implementation of these protocols is greatly simplified by their use.

This paper is organized as follows: section 2 provides a brief overview of the research context in which the tracker has been developed. In section 3 the Service Tracker framework and the interface are described, while section 4 gives the details on the Epidemic Service Tracker. Section 5 contains some implementation details and thoughts on future work.

An earlier, preliminary description of the Service Tracker was distributed under the title *ws-membership*.

2 The Obduro Project

In the Obduro[1] Project we are applying our experiences in building reliable distributed systems to the services oriented architectures. The web service architectures provide us with an opportunity to design robust distributed systems that are highly scalable. We have long been arguing that the scalability of distributed systems has been severely hindered by hiding the distributed aspects of the services behind a transparent single address-space interface. The "software as a *distributed* service" approach will allows us to build distributed systems more reliably than in the past [18].

In the recent years our research group has shifted its focus to the reliability problems triggered by the increasing scale of distributed systems and the failure of traditional systems to provide robust operation under influence of scale. In the Obduro project we will continue to exploit technologies such as epidemic communication and state-management to provide frameworks for building scalable, autonomous web services in federated setting.

The project focuses on deliverables in 4 areas:

1. Development of advanced distributed services in the context of the web services Coordination framework. These services will include a failure management service, a consensus service and a lightweight distributed state-sharing engine.
2. Development of high-performance server technology for web service routing, routing to transport mapping, content based routing and service selection. This technology specifically targets high-performance enterprise cluster environments.
3. Integration of reliability and other advanced distributed systems services into coordination and choreography engines.
4. Development of a framework for global event dissemination and event management for real-time enterprise applications. This development will particularly focus on global scalable reliable event systems based on a publish/subscribe paradigm. [17]

The work described in this paper comes out of the first area. The Service Tracker is an essential tool in managing service availability and is a building block for other, more complex distributed services.

3 A Service Tracking Framework

Based on our experiences with building a variety of distributed systems and applications, we realize that there is no single service tracking or membership service implementation that serves the needs of all (reliable) distributed systems. Some may require high accuracy while others need aggressive fail-over, where again others may require distributed agreement before failures are reported.

[1] Obduro is latin for *to last long, to be hard, persist, endure, last, hold out*

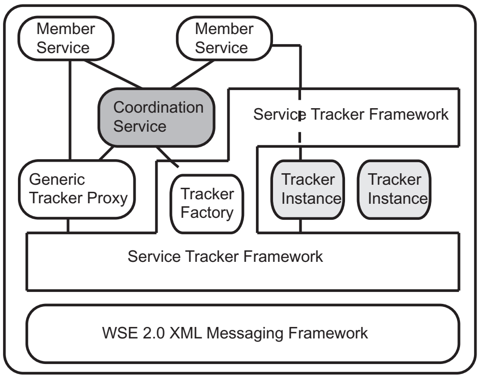

Fig. 1. Some of the components involved in Service Tracking

World wide failure detection requires different protocols than clustered systems in a datacenter. In the Obduro project we have implemented a prototype base on epidemic techniques that is particularly well suited for environments where robustness and scalability under adverse conditions is important. We expect to develop additional prototypes that target will different deployments.

These different implementations are developed using the Service Tracking Framework and have a single Service Tracking interface in common which services use to register themselves and service monitors can use to track the registered member services. The Service Tracking Framework, which relies on the Obduro Coordination Toolkit, provides the membership developer with basic classes that abstract the interaction of the tracking engine with member services and proxies.

The Service Tracking Interface is designed according the Activation/Registration pattern prescribed by the *ws-coordination* specification. The main task for the Coordination Service in this setting is to route the activation and registration requests to the appropriate component factory or instance multiplexer based on *CoordinationType* and *CoordinationContext* which are specified as parameters in the service requests

There are five roles modeled in the Service Tracking system:

– *Coordination Service.* Receives the activation and registration requests and routes these to the Service Tracking Framework. Its functionality is described in the *ws-coordination* specification.
– *Service Tracker.* Provides failure detection of registered web services and disseminates membership information. The actual implementation depends on the tracker type selected.
– *Member Service.* A software component that has registered itself for failure detection, either directly with a Service Tracker instance or through a Tracker Proxy. If the service registers itself with Service Tracker or with the Generic Tracker Proxy, it must implement its part of the tracker interrogation protocol.

- *Tracker Proxy.* A software component that is interposed between a member service and the Service Tracker for reasons of efficiency or accuracy. The proxy can be a dedicated component that uses specialized techniques to monitor member services such as a process monitor or an application server management module [19]. A generic version of the Tracker Proxy is implemented in the Service Tracker Framework for use by the Coordination Service in case a registration request is accepted for which no earlier activation action was taken at this Coordinator.
- *Services Monitor.* This service registers itself with the Service Tracker to receive changes to the membership state. While traditionally this functionality was combined with the Member Service role, e.g. registering as a member meant receiving membership updates; in this setting this functionality is decoupled. This allows for example the failure monitor of a BPEL4WS activity to register for membership updates with being part of the membership itself. In this usage example the individual member services have no use for the membership information as any compensating actions for service failure are initiated by the failure monitor.

The process of activation and registration will make the relation between these roles clearer.

3.1 Activation and Registration

The Coordination Service provides a 2-step access to Service Tracking. In the first step a participant in the activity, such as a member service or a activity monitor, *activates* the tracking system by sending a *CreateCoordinationContext* request to the Coordination service with the CoordinationType set to the following uri:

http://ws.cs.cornell.edu/schemas/2002/11/cucsts.

This will instantiate a Service Tracker at the coordination site using optional configuration parameters in the extensibility elements of the request. The operation will return a *CoordinationContext* to the requestor which includes information for other Service Tracker instances to join in the protocol.

If the activation is requested with an existing *CoordinationContext* as a parameter, the Coordination Service actives the Service Tracker, which joins the other Tracker instances activated with the same context. For example the Epidemic Service Tracker uses the information in the context to contact an already activated Service Tracker instance to bootstrap the current membership list. The new instance will add information about itself to the context, such that future activations at other coordinators have alternative Service Trackers to bootstrap from.

When the Service Tracker instances are activated at the Coordinators, the set of Trackers is established that will monitor each other. However if no *registration* request reaches a particular Tracker instance within the time-to-live period specified in the context, that instance will de-activate and will voluntarily leave the Tracker membership. The service membership list remains empty until Member Service instances are registered.

A service instance will request to be added to the membership through the *RequestMemberService* action. Parameters to the request include a URI uniquely identifying the Member Service instance, a *CoordinationContext* that represents the set of Service Trackers collaborating in the membership tracking and a port-reference on which the service is willing to accept *MemberProbe* interrogation messages. In the response to the request the service will receive a port-reference at the Service Tracker to which it can send its *MemberAlive* interrogation responses and the *MemberLeaves* notification when the service wants to exit the membership gracefully.

A second possibility for a service to be added to the membership is through interposing a Tracker Proxy. As described earlier such a proxy can be a dedicated software module that uses specialized techniques to monitor the service. Also a generic Tracker proxy is interposed by a Coordination service in response to a *RequestMemberService* request when no Service Tracker was previously activated at this Coordinator for the specified CoordinationContext.

Proxies register with a Service Tracker instance through the *RegisterTracker-Proxy* request which contains a port-reference at the proxy to which *ProxyProbe* interrogation messages will be send and the unique id the service provided to the proxy. A proxy can register itself multiple times for different services using the same port reference. The response to the proxy contains the port-reference at the Service Tracker to which the proxy can send its *ProxyAlive* message as well as *MemberLeaves* and *MemberFailed* messages when a Member Service connected to a proxy fails. The proxy interrogation sequence is used to keep track of the health of the proxy service, if the proxy is determined to have failed; all associated services will be marked failed in the membership. How a dedicated proxy determines the health of a service is outside of the scope of this paper, but the generic proxy uses the regular *MemberProbe/MemberAlive* interrogation sequence.

A Services Monitor registers using the *RegisterServicesMonitor* request, which includes a port-reference on which the monitor wants to receive the *MembershipUpdate* messages. A membership update includes at minimum a list of the currently active Member Services, but can be extended with information on Members Services that have left gracefully or recently joined, that have failed or are suspected to have failed.

To ensure that no false information is spread about the health of the Member Services, the Service Tracker, proxies, members and monitors all use the ws-security services to sign their messages.

4 The Epidemic Service Tracker

The basic Service Tracking engine implemented for the Obduro Project is based on a gossip-style failure detector, which was first described in [13]. This proba-

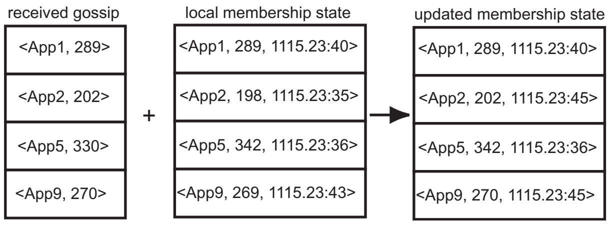

Fig. 2. Example heartbeat counter state merge

bilistic approach to failure detection was the basis for the design of the membership services in the Galaxy federated cluster management system [16] and in Astrolabe [14] which is an ultra-scalable state maintenance system. This prototype is based on experiences with the failure detection modules in those systems.

The Service Tracker is based on epidemic state maintenance techniques which provide an excellent foundation for constructing loosely coupled, a-synchronous, autonomous distributed components. The guarantees offered by the service are of the 'eventual consistency' category. For a member tracking service this means that if at any given moment one takes a snapshot of the overall system not all participants may have an identical membership list, but that eventually, if no other external events happen (members join, leave or fail), all participants will have the same state.

Additional advantages of using epidemic techniques for the tracking service are

– The strong mathematical underpinnings of the epidemic techniques allows us to calculate the probability of making a mistake (e.g. declaring a participant failed while it has not) based on the chosen communication parameters. One can adjust these parameters at cost of accuracy or increased communication
– The communication techniques used to exchange membership state among the participants are *highly robust*. Message loss or participant failures have only limited impact on the performance, and only then in terms of latency until all participants have received the updated state. When using epidemic techniques, it is not necessary to receive state directly from a participant, but this information can arrived through other participants.
– The membership information exchange between participants is implemented as a purely *a-synchronous* action. A participant will send a digest of its state to another participant without expecting that participant to take any action or to provide a response. The communication is really of the form *fire-and-forget*, which contributes to the scalability of the service architecture. In large scale federation settings one can expect very heterogeneous participant capabilities, in terms of processing and network access, and relaxing the synchronous nature of the system will enable us to deal with this heterogeneity more efficiently.
– Participants are able to reach decisions *autonomously* about failures of other participants. No additional communication such as an agreement protocol is

needed as the decision can be based on a combination of local timestamps and the configuration parameter of the epidemic protocol. Basically if the state related to a remote participant has not changed for a time-period based on the gossip interval and number of members, which a certain probability one can state that none of the participants has received any communication from this participant.

These robustness and scalability properties make this approach to failure detection and membership management attractive to use in a large federated environments.

There are some disadvantages to using epidemic failure detection. The protocol in itself can become inefficient, as the size of the messages exchanged grows with the number of participants. A number of optimizations have been developed using adaptive and hierarchical techniques, allowing the protocol to scale better. The prototype implementation targets small to medium membership sizes up to 50 – 100 participants, for which it is not necessary to implement these optimizations. A second disadvantage is that the protocol does not deal very well with massive concurrent participant failures, which temporarily influences the accuracy of the detection mechanisms, although we have not seen any other light-weight membership modules that do much better.

The detection of failed participants is very accurate, but is often configured rather conservative. In other settings where we have used epidemic membership management and where a more aggressive approach was needed we added a layer of interrogation style probing modules to enabling early suspicions of failures.

4.1 Operational Details

The Epidemic Service Tracker (EST) is based on the principles developed in the context of the Xerox Clearinghouse project [7]. In epidemic protocols a participant forwards its local information to randomly chosen participants. In Clearinghouse this protocol was used to resolve inconsistencies between distributed directories, in EST the protocol is used to find out which participants are still 'gossiping'.

The basic operation of the protocol is rather simple: each participant maintains a list of known peers, including itself, with each entry at least holding the identification of the remote participant, a timestamp and a single integer dubbed the *heartbeat counter*. Periodically, based on protocol configuration, the participant will increment its own heartbeat counter and randomly select a peer from the membership list to send a message containing the all the <address, heartbeat> tuples it has in its membership list. Upon receipt of a message, a participant will merge the information from the message with its local membership list by adapting those tuples that have the highest heartbeat counter values. For each member for which it adopted a new heartbeat value, it will update the timestamp value in the entry. See Figure 2 for an example.

If a participant's entry has not been updated based on the configured failure period, it is declared to have failed and associated monitors are notified. The

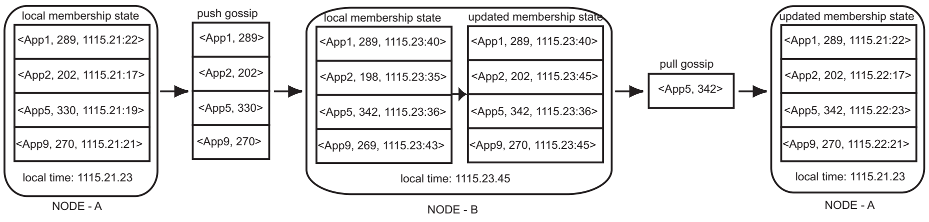

Fig. 3. Example *push-pull* style epidemic exchange of the heartbeat counters. Node-A gossips to Node-B a digest of its membership state. Node B, after merging the update with its local state returns a message to Node-A with the state that it knows is newer. As shown in the figure there is no need for the local clocks to be synchronized, all actions based on time are on local clock values.

failure period is selected such that the probability of an erroneous detection is below a certain threshold, and is directly related to the gossip communication interval. For a detailed analysis of the failure detection protocol and the relation between the probability of a mistake and the detection time see [13].

The EST prototype departs from the original model by implementing a *push-pull* model for its communication instead of the *push* used in the original design. In this model the receiving participant will, if it detects that is has entries in its membership table that are newer (e.g. have higher heartbeat counter values), send a message back to the original sender with those tuples for which it has newer values. The push-pull model has a dramatically better dissemination performance than the push model, especially in those cases where the local state at the participants change frequently, which is the case in our system. See figure 3 for an example of a push-pull operation.

4.2 Service Membership

The membership protocol operates between instances of the Service Tracker, and the failure detection described in the previous section detects failures of other Service Tracker instances. The membership however describes Member Services, and multiple such services can be registered with a single Service Tracker instance.

To implement the Member Service membership, the membership list is extended to include a list of Member Services registered with each Service Tracker instance, and this information is included in the gossip messages between the Service Trackers. Each Tracker instance can thus construct the complete list of all Member Services associated with the CoordinationContext. If a Service Tracker instance fails, all the associated Member Services are marked as failed.

If a Service Tracker determines that a Member Service has failed, either through the repeated failure of an interrogation sequence or through the *MemberFailed* indication of a Tracker Proxy, the service is marked as failed in the membership list. Other instances will receive this information through the epidemic spread of the membership information.

The information per participant send in the gossip messages is organized in five sets of Member Services:

1. Members. This is the list of the Member Service URIs that are registered and are active. This information set includes a logical timestamp (the value of the local heartbeat counter) indicating when the set was last updated.
2. *Joined.* A list of Member Services that have recently registered, with each the logical timestamp of the moment of registration. These members also appear in the *Members* set, but are included separately to ease the parsing of large membership lists. Members are removed from this list based on the epidemic protocol configuration, and all the participants will have seen the join with high probability.
3. *Left.* When a Member Service gracefully exits, it should send a *MemberLeaves* indication to the Service Tracker it has registered with. This will remove the member from the *Members* list and places it in the *Left* set, annotated with the logical timestamp.
4. *Failed.* After a member has been detected as failed it is removed from the *Members* set and placed in this set, annotated with the logical timestamp.
5. *Suspected.* An option at Activation time is to specify a threshold that would mark a member as suspected, before it is marked failed. As soon a Member Service or a Service Tracker is suspected to have failed; the related members can be placed in the *Suspected* set without that they are removed form the *Members* list. This allows the protocol to be configured to be very conservative in marking members as failed, but to be more aggressive in indicating whether a member is problematic.

4.3 Optimizing Concurrent Activations

The implementation of the Epidemic Service Tracker optimizes the case of multiple activations of the protocol. If multiple protocol instances are created at a Coordination Service instance (e.g. they have a different CoordinationContext), and there is an overlap between the participants in the protocol, the different instances will share a common epidemic communication engine. This will ensure that no unnecessary communication will take place, by avoiding duplicate gossiping. It also allows smaller member groups to benefit from the increased robustness of using a larger number of epidemic communication engines.

A second optimization implemented for tracking large but stable service groups is to gossip only a digest of the membership information instead of the complete membership lists. In this case a participant will only gossip about last logical timestamp the *Members* list was updated. If a participant receives a message with a newer logical timestamp, it can request a full membership from the sender of the gossip message or from the participant for which the updated timestamp was received.

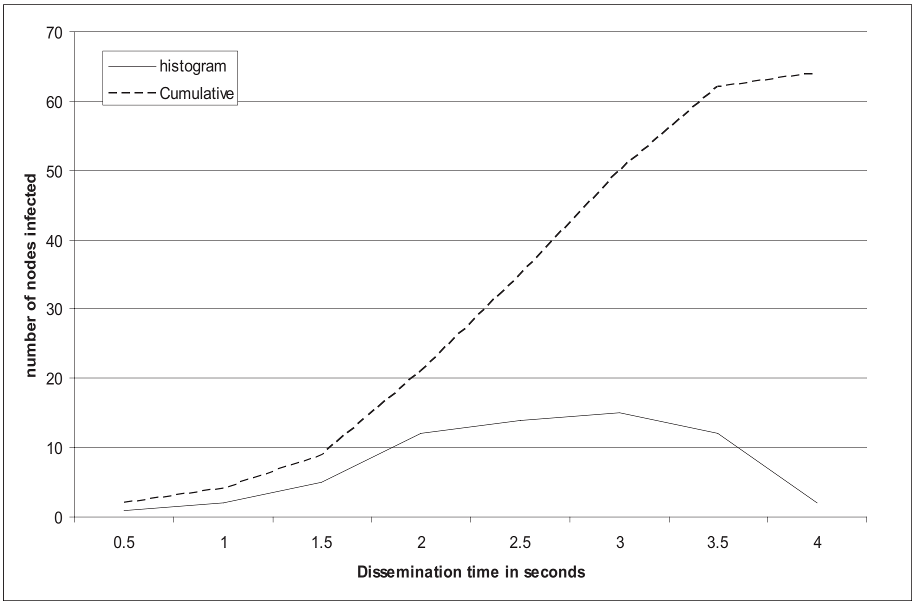

Fig. 4. Dissemination of membership information using epidemic techniques with 64 nodes with a 1 second gossip interval

5 Implemenation

The Coordination Service and Service Tracker Framework, and the Epidemic Service Tracker are implemented using functionality from Obduro Toolkit which in its part is based on the Microsoft WSE 2.0 XML Messaging Framework. A simple process monitor implementing the Tracking Proxy protocol has also been implemented.

The Service Tracker deploys 3 types of failure detection:

1. *Member Services* - through the *MemberProbe* interrogation protocol.
2. *Tracking Proxies* – through the *ProxyProbe* interrogation protocol.
3. *Service Trackers* – through the epidemic protocols.

It is a best practice to run as few remote Member and Proxy probes as possible. Running an instance of the Service Tracker on the same node as the Member Services or the Tracker Proxy will allow the probe protocols not to be subjected to network message loss, avoiding the need for a reliable transport, which improves the efficiency and accuracy of the probe protocol. The robustness of the inter- Service Tracker failure detection is not impacted by moderate message loss rate or even transient high loss rates.

In general a Service Tracker will use a separate dissemination channel to send membership updates to other participants in the membership protocol. In the case of the Epidemic Service Tracker we have chosen to include this information in the gossip messages. The advantages of this choice are that there is no need for additional communication primitives which in general needs to be a form of a reliable multicast protocol, or a brute force flooding technique. Adding a scalable reliable multicast would increase the complexity of the system while reliable flooding has scalability limitations. The epidemic dissemination of the membership information has the best of both worlds, it provide a flooding of the information, but in a very scalable, controlled and robust manner.

A disadvantage of the choice to piggy back the information on the gossiping of the heartbeats is that it disseminates rather slowly throughout the network. As can be seen in figure 4, in a setup with 64 instances of the Service Tracker it takes more than 4 seconds for the information to spread to all the nodes if these nodes only gossip once per second.

6 Related Work

We are not aware of any tracking or membership service, in research or production, which particularly targets long running business activities in service oriented architectures. We are also not aware of any public implementation of the Coordination Service/Framework that allows for dynamic management of new Coordination types.

There is a large body of work on failure detectors for different types of distributed systems, from reliable process group communication to large scale grids and from real-time factory floor settings to multi-level failure detection for clusters [4,6,8,9,10,13,14,15,19]. We expect that some of those implementations will also find their application in the web services world and we hope that our Coordination Serviceand Service Tracker Frameworks can be used to simplify the implementation and use of those systems.

An earlier preliminary description of the Service Tracker was distributed under the title *ws-membership*.

7 Future Work

This is only the first step in the development of a range of technologies as noted in section 2. One of the first follow-up tasks is to investigate how we can integrate the Service Tracker into the execution environment of system that use BPEL4WS as a driver

The software distribution of the Obduro Toolkit, the Coordination Service, the Service Tracker Framework and the Epidemic Service Tracker is slated for pubic availability in the fall of 2003.

Acknowledgements. This work is supported by a grant from Microsoft Research and by funding from DARPA under AFRL contract F30602-99-1-0532 and AFOSR/MURI contract F49620-02-1-0233.

References

1. Birman, K., and van Renesse, R., Software for reliable networks, Scientific American, 274, 5, pp. 64–69, May 1996.
2. Cabrera, F., Copeland, G.,Cox, B., Freund, T., Klein, J., Storey, T., Thatte, S., Web Services Transaction (ws-transaction), 2002, http://www.ibm.com/developerworks/library/ws-transpec/
3. Cabrera, F., Copeland, Freund, T., Klein, J.,Langworthy, D., Orchard, D., Shewchuk, J., and Storey, T., Web Services Coordination (ws-coordination), 2002, http://www.ibm.com/developerworks/library/ws-coor/
4. Chandra T.D. and Toueg S., Unreliable failure detectors for reliable distributed systems. Journal of the ACM , 43(2), pp:225–267, March 1996.
5. Chandra, T.D., Hadzilacos, V., and Toueg, S., The Weakest Failure Detector for Solving Consensus Proceedings pf the. 11th annual ACM Symposium on Principles of Distributed Computing, pages 147–158, 1992
6. Das, A., Gupta, I., Motivala, A., SWIM: Scalable Weakly-consistent Infection-style Process Group Membership Protocol, in Proceedings. of The International Conference on Dependable Systems and Networks (DSN 02), Washington DC, pp. 303–312, June, 2002
7. Demers, D. Greene, C. Hauser, W. Irish, and J. Larson. Epidemic algorithms for replicated database maintenance, in Proceedings. 6th Annual ACM Symp. Principles of Distributed Computing (PODC '87), pages 1–12, 1987
8. Felber, P., Defago, X., Guerraoui, R., and Oser, P., Failure detectors as first class objects, in Proceedings. of the 9th IEEE Int'l Symp. on Distributed Objects and Applications(DOA'99), pages 132–141, Sep. 1999.
9. Fetzer, C., Raynal, M., and Tronel, F., An adaptive failure detection protocol, in Proceedings. of the 8th IEEE Pacific Rim Symp. on Dependable Computing(PRDC-8), 2001.
10. Gupta, I., Chandra, T.D., and Goldszmidt, G., On Scalable and Efficient Distributed Failure Detectors, in Proceedings of the 20th Symposium on Principles of Distributed Computing (PODC 2001), Newport, RI, August, 2001.
11. Lamport, L., The Part-Time Parliament, ACM Transactions on Computer Systems 16, 2 (May 1998), 133–169.
12. Lampson B.W., How to build a highly available system using consensus, in Proceedings of 10th Int. Workshop on Distributed Algorithms, pp:9–11, Bologna, Italy, (October 1996).
13. Renesse, van R. Minsky, Y., and Hayden, M., A gossip-style failure detection service", in Proceedingc. of Middleware'98, pages 55–70. IFIP, September 1998.
14. Renesse, van R. Birman, K., and Vogels, W., Astrolabe, A Robust and Scalable Technology for DIstributed System Monitoring, Management and Data Mining ACM Tranactions on Computer Systems, Volume 21, Issue 2, pages 164–206, May 2003.
15. Stelling, P., Foster, I., Kesselman, C., Lee, C., and von Laszewski, G., A fault detection service for wide area distributed computations, in Proceedings. of the 7th IEEE Symp. On High Performance Distributed Computing, pages 268–278, July 1998.
16. Vogels W., and Dumitriu, D., An Overview of the Galaxy Management Framework for Scalable Enterprise Cluster Computing, in the Proceedings of the IEEE International Conference on Cluster Computing: Cluster-2000, Chemnitz, Germany, December 2000

17. Vogels, W., Technology Challenges for the Global Real-Time Enterprise, in the Proceedings of the International Workshop on Future Directions in Distributed Computing, Bertinoro, Italy, June 2002.
18. Vogels, W., van Renesse R., and Birman, K., Six Misconceptions about Reliable Distributed Computing, Proceedings of the 8th ACM SIGOPS European Workshop, Sintra, Portugal, September 1998.
19. Vogels, W., World-Wide Failures, in the Proceedings of the 1996 ACM SIGOPS Workshop, Connemora, Ireland, September 1996.