Ratatoskr: Wide-Area Actuator RPC over GridStat with Timeliness, Redundancy, and Safety

Erlend S. Viddal[*], Stian Abelsen[†], David Bakken[‡] and Carl Hauser
Washington State University
School of Electrical Engineering and Computer Science
Pullman, Washington, USA
{eviddal, stian.abelsen}@gmail.com, {bakken, chauser}@wsu.edu

Abstract

GridStat is a middleware framework designed to replace the power grids aging and inflexible data communications system. GridStat uses a specialization of the publish-subscribe paradigm, status dissemination, that takes advantage of the semantics of status data updates to provide data delivery with multiple dimensions of QoS semantics. While GridStat provides robust and timely support for data acquisition, the publish-subscribe architecture supports only one-way communication and provides semantics unsuitable for control communications.

In this paper we present Ratatoskr, an RPC mechanism that builds on the QoS semantics of GridStat are built upon to provide the timeliness required for power-grid operation. Additionally, user-defined pre- and post-condition expressions over GridStat status variables are built into call semantics. Pre-conditionscan abort an RPC call if the measured conditions dictate, while post-conditions provide additional physical assurance the call succeeded. The architecture of Ratatoskr is presented, along with results from an evaluation of a prototype implementation.

1. Introduction

Why grid communication today is inferior, [6]
Requirements for control traffic, [4]
GridStat data acquisition, [2]
Why GridStat needs control communication
RPC familiar semantics [3] Why CORBA would not do
Ratatoskr[1]

---

[*] Current affiliation: Simula Innovation, Oslo Norway

[†] Current affiliation: Eltek Valere, Drammen Norway

[‡] Contact author

[1] In Norse mythology, Ratatoskr is a squirrel climbing around the great world tree Yggdrasill, ferrying messages and gossip between the mythological creatures living in its branches and in particular insults between an eagle residing in the treecrown and a dragon gnawing on the roots.

Contributions:

- Design and implementation of a novel control scheme for an electrical power grid environment where remote procedure calls are transported over a QoS enabled one-way publish subscribe middleware network (GridStat).

- Design and implementation of three distinctive techniques for redundancy, offering a tradeoff between worst-case deadline, use of network resources and resiliency towards a variety of network failure categories. Applications are allowed fine control of redundancy semantics.

- Design and implementation of pre- and post- conditions mechanisms designed into RPC semantics provides additional functionality over application-level implementation and allows for a standardized mechanism for control signals between utilities.

- An experimental evaluation quantifying the tradeoffs between the redundancy techniques and their performance.

2 GridStat

The GridStat architecture is separated into two main subsystems, the *data plane*, a middleware databus where status updates supplied by publishers are forwarded to subscribers, and the *management plane*, a set of servers that manages system resources and organizes subscriptions by receiving subscription requests from subscribers and configuring the data plane towards forwarding accordingly. GridStat uses two kinds of communication traffic: *Data traffic* is always forwarded through the data plane message bus; *control traffic* between GridStat entities can be sent over any middleware control mechanism. The current implementation of GridStat uses CORBA and Ratatoskr as control message mechanisms.

Forwarding in the data plane is performed by *status routers*, middleware routers placed throughout a wide area network. Status routers form an overlay network by forwarding status events from router to router. The status routers retain implementations of all protocols used in the wide area network, and may function as bridges between the

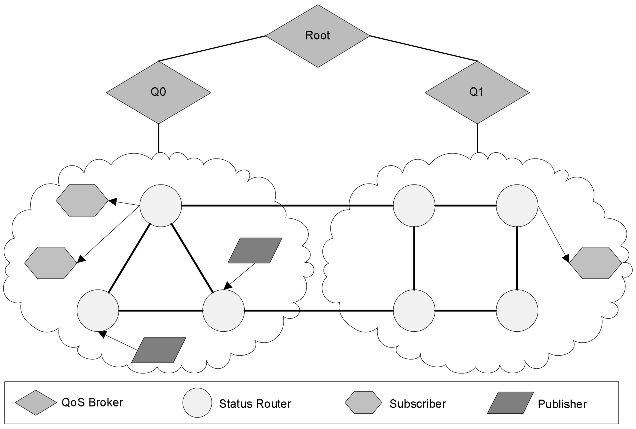

Figure 1. Example of a GridStat deployment

parts of the network using different networking technologies or with separate addressing spaces. Network connections in the data plane (from publishers and subscribers to status routers and between status routers) are represented as *event channels* that contain abstractions of data forwarding properties required for resource management. Each publisher and subscriber has event channels to one or more status routers.

Whereas the data plane has a flat organization, the management plane consists of a hierarchy of servers called *QoS brokers*. QoS brokers in the lowest level of the hierarchy are *leaf QoS brokers*, and are the only QoS brokers that directly communicate with entities in the data plane. QoS brokers above the leaf level are called internal QoS brokers and act as the sole *parent QoS Broker* of one or more *child QoS brokers*. All QoS brokers have a parent, with the exception of the *root QoS broker*, and leaf QoS brokers do not have child QoS brokers. Each QoS broker is associated with a set of entities in the data plane, the QoS broker's *cloud*. The data plane is divided up so each status router belongs to the cloud of exactly one leaf QoS broker. Status routers that have event channels to the same publisher or subscriber must be in the same cloud, and publishers and subscribers belong to the same cloud as their status routers. The clouds of internal QoS brokers are defined as the union of the clouds of their children, and thus the cloud of the root broker is all entities in the data plane. Entities are named according to their relationship to the management plane hierarchy. A GridStat element must have a name unique within the scope of its parent; its full name is the name within the scope with an added prefix of the parent's name. This hierarchy of clouds is meant to correspond to a natural organization of management domains in the power grid, such as levels of geographical areas. Figure 1 illustrates a GridStat system, with a hierarchy of QoS brokers managing status routers, subscribers and publishers divided into two leaf-level clouds.

As the data plane provides bounded delay and other QoS guarantees for subscription data, additional subscriptions must not overload network resources. The management plane administers the use of resources in the data plane, and so handles subscription requests. Subscription requests are made by the subscriber to its leaf QoS broker. If both the publisher and the subscriber of a new subscription are within the leaf level QoS broker's cloud, the leaf-level QoS broker is responsible for verifying that the connection will not overload network resources and update the status routers with the new subscription. If the publisher and subscribers are in different leaf-level clouds, the subscription request is propagated up in the hierarchy to the first QoS broker that has both within its cloud.

More details on the architecture and design of GridStat can be found in [5]

3 Ratatoskr

Ratatoskr uses a two-tier architecture. A communication module implements the 2WoPS protocol, providing timely, fault-tolerant two way communication over one-way GridStat paths; and an RPC module provides RPC server and client functionality. Separating the communication protocol into a separate module allows the communication module to be reused by other applications.

In [1], the 2WoPS protocol was utilized for control communication between GridStat QoS brokers. Figure 2 shows an overview of the modules of Ratatoskr (light shade), the GridStat modules used by Ratatoskr (dark shade), and other applications using the modules (white). The example shows the architecture stack for a control center and a substation. The main intended use of Ratatoskr is illustrated by the control center control system using Ratatoskr RPC to execute control operations on an actuator in the substation. Other uses of the 2WoPS protocol may be to transport legacy control messages to actuators if the actuator API remains to be fully implemented for Ratatoskr. The publisher and subscriber used by the 2WoPS protocol may have other uses, such as sending sensor data from the substation to the control center, or publishing reports of power-grid state aggregated in the control center to be used by protection schemes in the substation. Finally, while GridStat requires control of the underlying network resources, network technologies that manage resource use may reserve bandwidth for uses outside GridStat, such as transferring video feeds from surveillance cameras in the substation.

3.1. The 2WoPS protocol

2 pages To achieve a two-way communication style needed for RPC signaling, Ratatoskr uses two GridStat pub-sub connections. Each RPC host utilizes both a publisher and a subscriber - the publisher for sending data as published status variables and the subscriber for receiving data. Two hosts wishing to communicate each publish a new status variable and subscribe to the other hosts published variable. Data is sent over the connection by publishing status

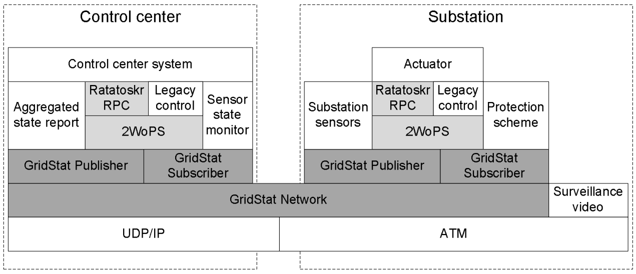

Figure 2. Ratatoskr Module Stack

events. It should be noted that as GridStat uses the status dissemination pub-sub paradigm for message forwarding, sending data in this manner has a very low routing overhead, as paths are established at connection setup time and forwarding ports are mapped to a simple connection identifier. Currently the 2WoPS protocol supports only communication between two hosts, but the GridStat network has excellent support for multicast of status events so future versions could add support one-to-many and group communication styles. For the remainder of the paper, the initiator of a connection will be referred to as a 2WoPS client and a receiving host the 2WoPS server, although in practice no distinction is made after connection setup.

The 2WoPS protocol was designed to draw upon the QoS guarantees provided by the GridStat publish subscribe paths to supply timely communication. Further, the protocol aims to provide a high degree of fault tolerance while retaining predictable delivery times. To this end, three redundancy techniques are employed to strengthen delivery guarantees:

- *Spatial redundancy:* One major feature of GridStat is that subscriptions may be routed through multiple disjoint network paths. Thus, if a status event sent over two redundant paths is corrupted along one path, successful delivery is still possible along the secondary path. The 2WoPS protocol exposes the number of disjoint paths used for the paths between the 2WoPS client and server and for the return path during setup connection time, and filters any excess status events upon receival. For a connection in which both directions employ n disjoint paths, we say that the connection has a *spatial redundancy level* of n. Status events sent over such a connection will tolerate $n-1$ losses, and are not affected by temporal concentration of errors. It should be noted that common mode failures thoughout the network, such as very high traffic levels during attacks or crisis situations may affect components involved in all redundant paths. The degree of spatial redundancy possible for a connection depends on the network topology as the network must support disjoint paths from the 2WoPS client to the 2WoPS server, but it is expected that the high reliability requirements of a critical infrastructure as the power-grid will justify the expense of building a network with a high degree of redundancy. Further, spatial redundancy will incur some delay overhead because of routing complexity and, also depending on the network topology, as the delay-optimal route might be incompatible with the number of disjoint paths. It should be noted that the process of allocating redundant paths through the network is more complex than allocating a single path, and so spatial redundancy induces overhead to the network management. GridStat supports delay guarantees also for paths redundant to the best-delay route (the *primary path*), and so a connection employing n spatial redundancy will provide delay guarantees even in the face of up to $n-1$ losses. The current implementation of GridStat supports only up to two redundant paths, and does not allow redundant paths between leaf-level clouds, but work is done to eliminate these limitations.

- *Temporal redundancy:* While spatial redundancy provides efficient protection from a range of network failures, the network costs and delay properties of a large number of paths could prove prohibitive in cases where a very high degree of fault tolerance is required or where the network topology is unsuited for spatial redundancy. Ratatoskr offers temporal redundancy by allowing each data unit sent by the 2WoPS protocol to be repeatedly published as several status events carrying the same data. A data unit published n times has a *temporal redundancy level* of n and tolerates $n-1$ losses. The technique consumes n times more bandwidth than regular sends, but has no delay overhead or additional setup costs. Temporal redundancy does not provide the same level of protection as spatial redundancy, as network losses may be temporally concentrated. For example, network congestion will often lead to periods of high loss rates when a router's buffer for an outgoing link is full or maintenance on a router might disable all connecting links for several minutes. To add to this, by sending several copies of the same 2WoPS packet in a short span, the extra bandwidth used might add to existing congestion. The 2WoPS protocol allows applications to specify a delay between sending each temporally redundant copy of a 2WoPS packet, and the degree of temporal redundancy may be specified for each send.

- *ACK/resend*: While spatial and temporal redundancy provides a high degree of fault tolerance, they do not provide any feedback to the application about successful delivery, and will not provide protection from failures that affect the whole network, such as periods of heavy congestion. Ratatoskr also allows a third technique for network redundancy by having message receivers ACK successfully sent 2WoPS messages, and allowing the user to specify that messages that do not produce an ACK should be resent up to n times. A message that is set to be resent up to n times with missing ACKs has an *ACK/retry level* of n. Successfully received ACKs are reported back to the server, while messages that do not receive an ACK even after n resends are reported with *unknown delivery* status.

3.1.1 Combining redundancy techniques

Redundancy measures can be combined as seen fit by the user. Temporal and spatial redundancy measures are cumulative and affect all 2WoPS packets, including ACKs. For example, if a sending a 2WoPS packet with 3 temporally redundant sends and 4 ACK/resends over a connection with 2 spatially redundant paths in each direction, three status events containing copies of the message packet will be sent. At the entry-point router, each of the status events will be forwarded to the first routers of the redundant paths, and, assuming no network failures, six status events containing the same 2WoPS packet will arrive at the receiver. The receiver will similarly send a single ACK 2WoPS packet, which will be sent in three temporally redundant status events, which again will be copied onto the redundant paths. If all ACKs are lost in the network, the sender will resend three new status events containing the message, and so on. This gives application designers the ability to tailor a connection to the exact needs of the application, allowing use of high spatial and temporal redundancy where a low delay is required, or relying on ACK/resend for redundancy for less delay-sensitive applications or where bandwidth is scarce. Combining redundancy techniques in this manner gives a total delivery guarantee based on the cumulative worst cases of the redundancy techniques. The last packet of a message with temporal and spatial redundancy will have a deadline of the last temporally redundant packet along the worst of the redundant paths. This overhead will be doubled for messages with ACK/retry enabled because of the ACK.

4 The Ratatoskr RPC mechanism

Ratatoskr RPC is a remote procedure call protocol for power-grid control communication built on top of the 2WoPS protocol. The primary application for Ratatoskr is remote operation of power-grid actuators, either to a gateway interfacing multiple devices or by directly controlling intelligent electronical devices (IEDs) with remote interfaces embedded in the actuator itself. The current grid communication system is unable to support developments in the power-grid stress levels and the security threat picture, and proposed solutions require real-time, reliable control [4].

Ratatoskr draws extensively on the features provided by the 2WoPS protocol. Delivery guarantees for calls is achieved using GridStat's QoS enabled network communications, and fault-tolerance is provided through the redundancy techniques found in the 2WoPS protocol. As the use of redundancy trades off network resources, or worst case delay in the case of ACK/retry, against safety, applications designers are allowed detailed control over the level techniques used for redundancy.

In addition to increased safety through fault-tolerance, pre- and post-conditions on calls are built into the RPC semantics. Pre-conditions are conditional expressions over GridStat variables that are evaluated before execution of an RPC call, and if the expression is negative, the call is negated. Post-conditions are similar expressions evaluated after the call has executed, and negative results are reported back to the application, allowing evaluations of operation outcome.

Ratatoskr does currently not provide the security features required for a deployment in a critical infrastructure such as the power grid, and assumes no Byzantine behavior. Further, the prototype was implemented in Java, and the communication primitives does not accommodate platform independence. Finally, handling client and server failure is left for future work. This affects RPC failure semantics[12] in that only at-most once failure semantics are available. It should be noted that as the number of ACK/retries are customizable, the at-most once semantics found in Ratatoskr are more flexible than the traditional understanding, i.e. maybe once semantics may be achieved by setting the number of retries for a call to zero. If at-least once semantics or some form of exactly-once semantics should be implemented for Ratatoskr, similar flexibility would arise. Further information on the properties of Ratatoskr fault-semantics may be found in [13]

4.1 Pre-and Post Conditions

For many power-grid control-operations, placing pre-conditions on the execution may help in preserving safety in face of unwarranted situations such as power-grid anomalies or unexpected mechanical operation. Because of the distances involved, communication must be expected to suffer from bandwidth and delay limitations. Thus one may assume that the client-side information about server state is limited and not fully updated, and following this, pre-conditions should be placed on the server-side of the call. Such conditions could be placed in application code using RPC exceptions, but this could lead to variations in semantics between vendor implementations.

Ratatoskr incorporates pre-conditions in call semantics. This gives standardized predicate semantics, easing interoperability between equipment vendors and inter-utility communication.

Examples of pre-conditions in power-grid operation may be:

- Isolators are actuators that connect and disconnect de-energized power circuits. A precondition could be to verify that a line is de-energized before attempting isolation.

- High voltage equipment carries with it electrocution hazard, and another pre-condition could be to verify that no manned maintenance is scheduled at a field site when performing operations that might place maintenance personnel in danger.

Ratatoskr further allows application designers to use the same predicate modules used for pre-conditions for placing post-conditions on calls. Power-grid operations are complex, and actuator operations may give unexpected results in face of situations such as mechanical malfunctions or operator overrides. Server-side post-conditions will be able to utilize the rich information environment local to the substation for analyzing the physical outcome of an execution and return only a brief report to the client. Thus client applications will be able to review the results of calls without having to retrieve large amounts of data from the substation. By allowing a delay before the post-condition is evaluated, the effect of the operation is allowed to stabilize. By designing the post-conditions into the RPC semantics, the result of the post-condition is transferred to the client as a separate send, without affecting the delay of the RPC call itself.

Examples of post-conditions in power-grid operation may be:

- Load tap changer are components of certain transformers that allow adjustment to voltage output during load. A post condition could be to verify the new voltage level after load tap changer operations, or even generate a status report from all connected devices and send it back to the client.

- Transformer protection is a scheme to detect internal faults in a transformer and isolate it by braking all connected lines if a fault is detected. A post-condition could be to trigger a transformer protection scheme after all transformer operations.

5 Evaluation

Ratatoskr is evaluated with respect to performance in face of network faults. The purpose is to understand the efficiency of the implemented fault tolerance techniques on RPCs over a faulty network, not to evaluate the performance of the prototype implementation, as a real-time implementation is outside the scope of the prototype design.

5.1 Evaluation Procedure

Evaluations were performed by connecting Ratatoskr client and server processes to a small GridStat network and commencing a number of RPC calls from the client to the server. To introduce network faults, event channels between status routers were routed through a network link emulator introducing delay and errors. Links going through the link emulator are referred to as *emulated links*. This setup tries to emulate control traffic over a wide area network of status routers, where both the client and server are either connected to their entry-point SR by LAN or running as processes on the same computer. No delay or loss is induced on the link between Ratatoskr peers and their entry-point SRs. This is based on the expectation that a deployment of GridStat will include a wide deployment of status routers throughout the power grid to provide a high degree of network redundancy, which makes it likely that sites using RPC also contain a status router.

5.1.1 Evaluation Topology

The topology of the evaluation setup is shown in figure 3. 13 status routers form two paths between the client and the server, one of 7 links and one of 6. This is to allow for the fact that when using spatial redundancy additional paths may often be longer than a single best path. The current implementation of GridStat does not allow more than two redundant paths. No additional links and status routers outside the two paths were employed as routing between the same two peers in a static network (as the current version of GridStat is) will result in the same path no matter the errors.

5.2 Evaluation Setup

5.2.1 Processes

The processes used for testing were:

- 13 GridStat status routers
- 1 GridStat leaf QoS broker
- 1 evaluation program implementing a Ratatoskr client peer *(client)*
- 1 evaluation program implementing a Ratatoskr server peer *(server)*

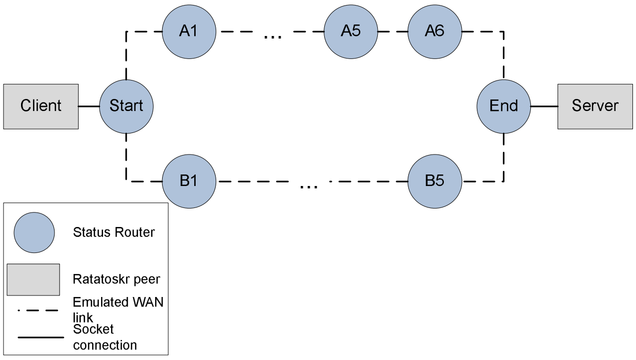

Figure 3. Evaluation Topology

- 1 Sun Java CORBA nameserver
- 1 Java program providing socket tunnels with loss and delay properties *(network link emulator)*

5.2.2 Hardware

The evaluation was tested on a single computer, running a Core2 duo 2.13GHz dual-core processor and 4 gigabytes of RAM. The operating system was Ubuntu linux, kernel 2.6.20-16 compiled with 1 millisecond kernel tick intervals and full kernel preemption. All evaluated programs were implemented in Java, compiled and run with sun java2SE 6 (version 1.6.0.00). All inter-process communication was done over operating system UDP sockets for GridStat data traffic and Sun's Java 2 Platform CORBA Package for control. All tests were performed in user mode with regular process priority and with the graphical operating system interface turned off.

5.2.3 Garbage Collection Handling

A mechanism was implemented to measure the impact of garbage collection and substract this from call results. The mechanism queries the Java `java.lang.management.GarbageCollectorMXBean` interface for recent garbage collection operations and their durations, and if gc operations have occured, substracts the gc operation duration from the end-to-end duration of the affected call.

5.3 Fault Models

GridStat uses multiple underlying network technologies, spans a wide area, and will sustain several usage patterns (usage patterns to this point includes rate based status updates and bursty RPC traffic). This makes for very complex behavior and it is difficult to provide a good fault model for GridStat without field testing. For this evaluation, two fault models were combined to account for the rich diversity of potential fault patterns in a GridStat deployment.

- *Omission-fault* - Each link is assigned a uniform probability of dropping each packet passing through it. The drop probability is the only variable for the omission-fault model. Each drop is completely isolated; no other link or later or earlier packet on the same link is affected by a drop. Omission-fault attempts to model temporally and spatially isolated drops in links where the no retry-upon-failure is attempted below the transport layer in the protocol stack. Examples of uniform drop rate errors are background noise or very short term physical interference in links causing packet data corruption: passing physical objects in the way of the beam of a microwave beam, bursts of electromagnetic noise from power anomalies in a substation wired with copper or frequency noise from grid devices in a broadband over power link.
- *Duration-fault* - Each link is in one of two states: disabled or enabled. If disabled, all packets passing through the link is dropped. If enabled, the operation of the link is not affected, and all packets pass through the link unless omission failures occur. Links are ordinarily enabled, except for 1-second periods of disable state. Disabled state periods occur by a Poisson process, where the average number of occurrences per second (λ) is the only variable for the duration-fault model. Duration-fault attempts to model transient failures in network components. Examples may include: router maintenance, fiber cuts or local powerouts. Such failures may certainly have a duration of well over a second, but a one second disable state duration is enough to notice the effects of duration failures on communication. It should be noted that dynamically routed networks will quickly adjust so signals will circumvent duration faults, even nearly instantaneously [7]. GridStat uses static routing, and so paths will not be adjusted even if faults are detected. A future solution for this in GridStat is for the QoS hierarchy to create new paths around failures, but this is an expensive operation when resource management calculations and communication to status routers is taken into concern, and must be expected to be time consuming. The primary mechanism for overcoming longer-term failures in GridStat is spatial redundancy.

5.4 Experiment Procedure

- For all experiments, the process running the Ratatoskr client also functioned as experiment coordinator.
- For each new experiment session, 10,000 RPC calls were made to warm up the system, and then one or more experiments were run sequentially.
- An experiment consists of 10,000 RPC calls with data gathered from each call.
- A new Ratatoskr connection was established for each experiment, and closed at the end of the experiment.
- All calls were made with an unlimited number of ACK/retries.
- Between calls, all links emulated in the link emulator was reset, specifically by setting the internal clock used for duration failure $2 * (1/\lambda)$ seconds into the future, in effect ending all previous disabled states and allowing new ones to arrive.

- A link delay of 1 millisecond was used unless specified otherwise in the experiment description.

- When temporal redundancy was used, a two millisecond delay between redundant sends was used.

- For all experiments without spatial redundancy, calls were made over a 6 emulated link path. For experiments using spatial redundancy, calls were made over one path of 6 emulated links and one path of 7 emulated links.

- The timeout for the ACK/retry technique had a base of 25 milliseconds, allowing 12 ms for transfer delay, 20 ms for garbage collection and 3 ms for overhead from the link emulator, GridStat rounting and Ratatoskr. Higher timeouts were assigned by Ratatoskr to sends with temporal or spatial redundancy due to the wait between redundant sends and the extra hop in the spatial paths. Specifically, using spatial redundancy added 2 ms to the timeout, and using temporal redundancy added 2 ms for each redundant send.

5.5 Result Data

The following data was extracted from each call:

- Time was measured from when the call was made until the result arrived (*end-to-end delay*).

- The number of timeouts experienced was recorded. This includes any timeouts the server experienced while attempting to send the result. Specifically, the number of timeouts for a call is the number of timeouts for the first time of a call to arrive at the server *not including timeouts introduced by missed ACKs*, and the number of timeouts for the result to reach the client the first time, again not including timeouts introduced by missing ACKs.

- Early success rate for experiments is defined as the number of calls with no retries divided by the total number of calls.

5.6 Resiliency of Temporal Redundancy

To evaluate the resiliency of the temporal redundancy mechanism, a series of experimental runs was performed with increasing degrees of temporal redundancy (1, 2, 4 and 8 sends). Each of the temporal redundancy levels was tested over a set of omission fault rates (1%, 2%, 4% and 8%) applied to all emulated links. Duration loss was omitted from the evaluation, and is addressed in a later experiment. The results are shown in figure 4, with corresponding expected results from analysis. The analysis combines the drop rates of each link into one drop rate for the 6-link path from client to server (f_{6-link}), and calculates the probability of all redundant packet not getting dropped ($s_{send}(1-f_{6-link})^n$ for temporal redundancy level n). For the end-to-end loss probablility, the probabilities for both the send and the return not getting dropped was calculated as s_{send}^2. The experimental results match the analysis very closely.

Figure 4 shows that two temporally redundant sends are not sufficient to entirely overcome a 1% loss rate, but with 99.2% early successes against 88.5% for the non-redundant calls it is still a good improvement. With 4 resends, 99.92% early successes are achieved at 4% failure rate. For 8 resends, 99.92% of calls were early successfull even at 8% loss rate, where the end-to-end early success rate without reliability was 36.5%. Even during periods of intensive network loss, critical applications where the extra bandwidth for temporal redundancy can be spared should be able to perform RPC calls with very few retries, given that the loss patterns accomodate temporal redundancy.

5.7 Resiliency of Spatial Redundancy

The efficiency of spatial redundancy was evaluated in a manner similar to temporal redundancy. Runs were performed with spatial redundancy enabled over increasingly higher occurrence frequencies of duration faults (1 second failure every 10000 seconds, 1s/1000s, 1s/500s, 1s/100s and 1s/50s). Values for expected results based on analysis are included in the diagram. An analysis similar to the analysis for temporal redundancy was made, simplifying the arrival rates of fault durations for links as fault percentages (1s/50s=2%, 1s/100s=1%...).

For the experiments, 100% early successes was achieved for both 1s/10000s and 1/1000s fault rates. As this is for 10,000 calls, this is encouraging as it satisfies at least a 99.99% end-to-end reliability requirement. For 1s/500s fault durations, 99.96% early success is achieved. The experment's results follow the analysis closely except for at 1s/50s fault rate. This could stem from the analysis simplifying the fault rate to a percentage, which might weaken the analysis for high fault rates.

5.8 Comparison to ACK/Retry-Only RPC

A final experiment was made to compare the performance of Ratatoskr to a traditional RPC call without other forms of reliability than ACK/retry, and to evaluate the effect of spatial and temporal redundancy on the failure models used. Here, performance refers to the ability to tolerate network faults and end-to-end delay over a faulty network. Experiment runs for no redundancy, 4 temporally redundant sends, spatial redundancy and the combination of the two

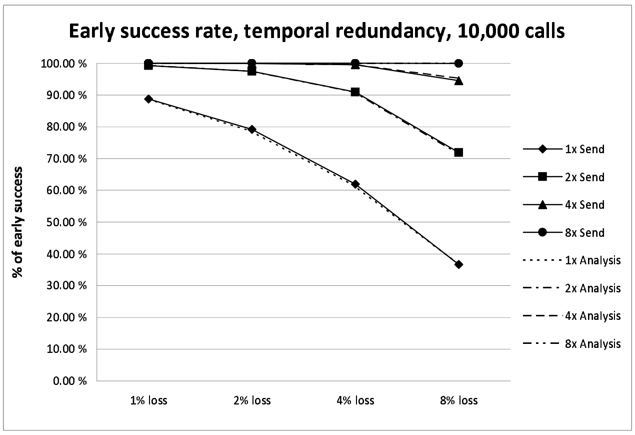

Figure 4. Early Success for Temporal Redundancy over Varying Omission Fault Rates

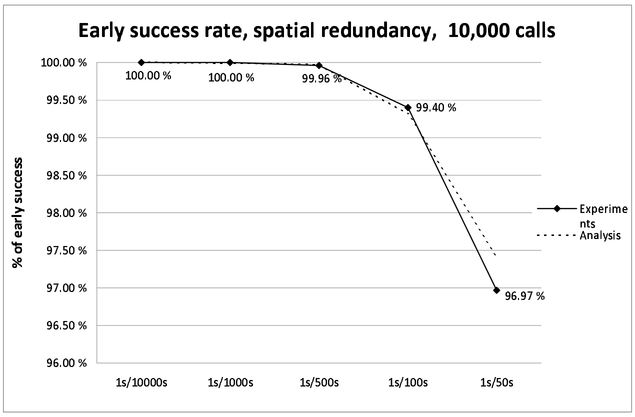

Figure 5. Early Success for Spatial Redundancy over Varying Duration Faults

were performed over 1% omission fault rate, 1 second fault duration every 10000 seconds, the combination of the two and no failures. The early success rates are shown in figure 6. The average duration with standard deviation for all redundancy levels with both fault models combined is shown in figure 7. Cumulative distributions of timeouts experienced before call success for all redundancy levels for the combination of fault models is shown in figure 8 (note that the percentage range on the graphs are different to allow visible details). The highest calltimes measured with the combination of losses are: 939 milliseconds for no redundancy, 91 ms for spatial redundancy, 977 ms for temporal redundancy and 24 ms for the combination of the redundancy techniques.

From figure 6, it is seen that for 1% omission loss, temporal redundancy experiences very few timeouts, but the spatial redundancy does not provide enough reliability for the fault rate level and 0.81% of the calls experience timeouts. Without any redundancy, less than 90% of the calls achieve early success, which makes for very unstable calltimes. The 1s/10000s duration fault setting does not affect the early success rate of any of the redundancy levels too much, while the combination of fault models has a pattern very similar to that of 1% omission failure.

In figure 7, the effects of duration loss becomes apparent. Temporal redundancy, which is ineffective against duration faults, has a considerably much higher standard deviation of end-to-end delays than spatial redundancy. Together with the low early success rate, this signifies that a few calls must retry several times before success is achieved, even with four redundant sends. The experiment run with no redundancy follows a similar pattern with high standard deviation of end-to-end delays, and also the average end-to-end delay is higher. This is most likely from the high percentage of calls that timed out. The end-to-end delay with both forms of redundancy retain the same average as spatial and temporal redundancy, and with a small standard deviation. The standard deviation patterns are reflected in figure 8. 99.85% all of the calls made with temporal redundancy incurred no timeouts, but the distribution has a long tail, and 0.05% of the calls incurred over 10 retries. The highest end-to-end delay measured for temporal redundancy was 977 milliseconds, and the highest number of retries was 22. The cumulative distribution for spatial redundancy compared to the temporal calls reveals that while a relatively high number of calls experience timeouts (over 0.8%), only a single call experiences the highest number of timeouts, 2, and 91 milliseconds was the highest measured end-to-end delay. With both redundancy techniques, no timeouts were experienced and the highest measured end-to-end delay was 24 milliseconds. Without redundancy, over 12% of the calls experienced timeouts and the highest number of timeouts was 25, with the measured end-to-end delay 939 milliseconds. This is a higher number of timeouts for a shorter measured end-to-end delay compared to the temporal redundancy call, as the timeout is set higher for temporal redundancy due to the 2 millisecond wait between redundant sends. From this we can conclude that even if a control mechanism over a single-path network uses a transport protocol with temporal redundancy, it may still experience very high call durations with longer-term failures in a single component on the path. Compared to RPC calls without the redundancy measures found in Ratatoskr, spatial redundancy greatly lowers the worst-case end-to-end call duration, temporal redundancy improves the average call duration considerably, and the combination improves reliability greatly.

6 Related work

Recent work on fault tolerant RPC mechanisms have been centered on host replication using distributed objects. As the distributed object interface is decoupled from the underlying implementation and environment, an object interface can be replicated into several implementations running in separate environments with minimum impact on observed behavior. Several CORBA implementations provide replicated objects, [10, 11, 8]. A replicated distributed object scheme, coupled with a real-time CORBA implementation, would provide timely delivery and fault-tolerance. While server replication mainly protects from server failure, protection from network failures would also be achieved to some degree. Depending on the redundancy scheme, sending to a group of replicated servers is achieved by multicasting from the client to each replicated host, and so the data is likely to go through several network paths. Such a scheme would still have to rely on an underlying network for network-level fault tolerance, and would not be able to reap the benefits of redundant path routing wtihout modification. Further, object replication has to rely on strong multicast guarantees for synchronization between replicas to achieve correct group communication, which gives high worst-case messaging rounds in face of communication failures and thus scales badly with geographical distance.

We are currently aware of only one RPC mechanism utilizing temporal and spatial redundancy: [9] provides an implementation of CORBA for embedded systems providing hooks for spatial and temporal redundancy, but does not provide an analysis or evaluation of the effects of these measures upon call reliability and effeciency.

7 Conclusion

Existing power-grid control mechanisms are inflexible, incompatible across vendors, and designed for highly centralized environments, and cannot provide for the next generation of power grid communication. While much work is

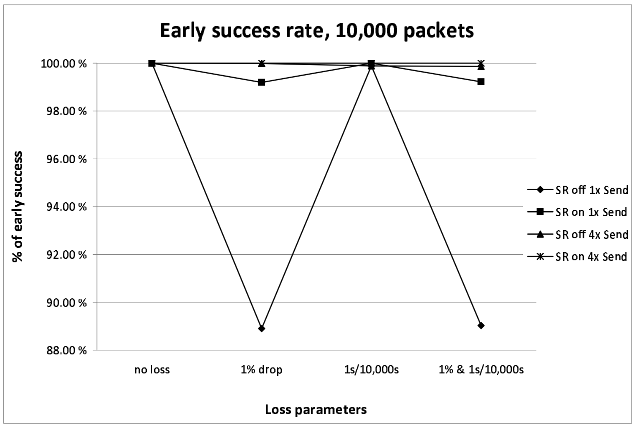

Figure 6. Early Success for Varying Redundancy and Loss

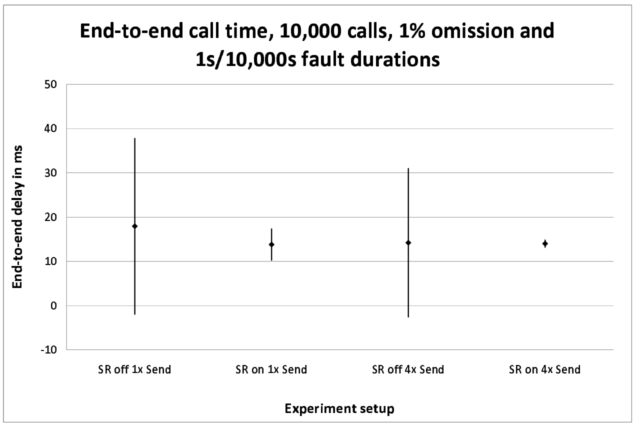

Figure 7. Average Calltimes for Various Redundancy with Full Loss

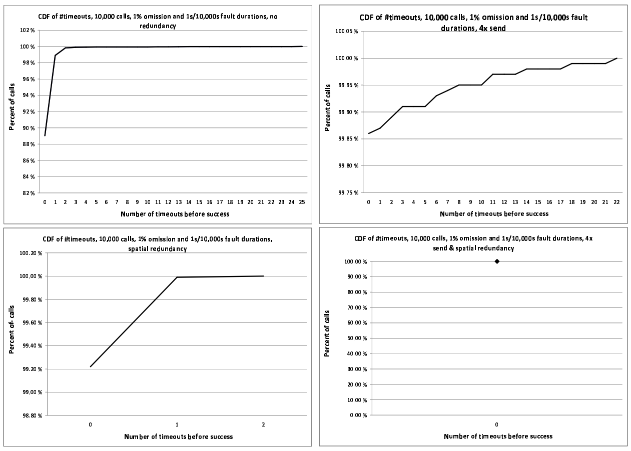

Figure 8. Cumulative distributions of number of timeouts per call

done in using web services and other existing control mechanisms, such control mechanisms built directly on top of existing network protocols will not overcome the incompatibility issues with proprietary protocols and may have to compromise safety and timeliness depending on the underlying network.

This paper provides the architecture and evaluation of Ratatoskr, an extension to the GridStat status dissemination system with the design and implementation of a timely and reliable remote procedure call mechanism running on top of two one-way GridStat subscription paths. This extension complements GridStat's capabilities as a middleware data acquisition system with the ability to send control messages to power-grid actuators.

Ratatoskr is split into two subsystems, a transport protocol for two-way communication over the GridStat network, the 2WoPS protocol, and an RPC mechanism built on top of the 2WoPS protocol, the Ratatoskr RPC mechanism. The 2WoPS protocol utilizes the QoS semantics provided by the GridStat network to offer maximum delivery delay guarantees and spatially redundant network paths for sends. Further, two additional redundancy techniques are used; temporally redundant sends and ACK/retires. While the 2WoPS protocol was implemented specifically for the RPC mechanism, it is also used in another GridStat project that gained from the timeliness and reliability provided by the protocol. The Ratatoskr RPC mechanism provides extensive customization of the redundancy levels for each call, providing a tradeoff space between timeliness, use of network resources, and reliability. Further, pre- and post conditions built into the call semantics provides additional safety mechanisms for application designers, and creates a building block for a uniform middleware safety framework for inter-vendor operations.

The evaluation explores the effectiveness of the redundancy techniques in the face of two fault-models, omission faults with uniform drop probability and durations of total link failure. The experiments show that both spatial and temporally redundant resends provide a polynomial reduction in end-to-end fault occurrence rates for omission faults, with a a temporal redundancy of two providing a square reduction, a temporal redundancy of three providing cube reduction and so on. The spatial redundancy shows a similar pattern with two redundant paths providing a square reduction, but as the current implementation of GridStat does not support more than two redundant paths, a conclusion on the developments of the pattern with more paths could not be established beyond mathematical models. Temporally redundant sends proved of little value in the face of link failure durations, but spatial redundancy provides a square reduction also here. The experimental results match the expected results from the analysis closely. A comparison between Ratatoskr and a simulated traditional RPC mechanism without temporal and spatial redundancy, shows that Ratatoskr profits from the redundancy with a lower average for end-to-end call times and a considerably tighter call time distribution.

Acknowledgements

Well deserved thanks goes to Erik Solum for helping review the research process and James Kusznir for advice and help with technicalities.

This research has been supported in part by grants CNS 05-24695 (CT-CS: Trustworthy Cyber Infrastructure for the Power Grid(TCIP)) and CCR-0326006 from the US National Science Foundation.

References

[1] S. F. Abelsen. Adaptive gridstat information flow mechanisms and management for power grid contingencies. Master's thesis, Washington State University, Pullman, Washington, USA, August 2007.

[2] D. E. Bakken, C. H. Hauser, H. Gjermundrød, and A. Bose. Towards more flexible and robust data delivery for monitoring and control of the electric power grid. Technical report, School of Electrical Engineering and Computer Science, Washington State University, 2007.

[3] A. D. Birrell and B. J. Nelson. Implementing remote procedure calls. *ACM Transactions on Computer Systems*, 2:39 59, 1984.

[4] EPRI/CEIDS. The integrated energy and communication systems architecture, volms i-iv, July 2004.

[5] K. H. Gjermundrød. *Flexible QoS-managed status dissemination middleware framework for the electric power grid*. PhD thesis, Washington State University, Pullman, Washington, USA, August 2006.

[6] C. Hauser, D. E. Bakken, and A. Bose. A failure to communicate: next generation communication requirements, technologies, and architecture for the electric power grid. *Power and Energy Magazine, IEEE*, 3:47–55, March-April 2005.

[7] S. Lee, Y. Yu, S. Nelakuditi, Z.-L. Zhang, and C.-N. Chuah. Proactive vs reactive approaches to failure resilient routing. *Twenty-Second Annual Joint Conference of the IEEE Computer and Communications Societies, IEEE INFOCOM*, 1:176 185, 2004.

[8] S. Maffeis. Adding group communication and fault-tolerance to corba. In *COOTS'95: Proceedings of the USENIX Conference on Object-Oriented Technologies on USENIX Conference on Object-Oriented Technologies (COOTS)*, pages 10–10, Berkeley, CA, USA, 1995. USENIX Association.

[9] A. D. McKinnon. *Supporting fine-grained configurability with multiple quality of service properties in middleware for embedded systems*. PhD thesis, Washington State University, Pullman, Washington, USA, December 2003.

[10] L. E. Moser, P. M. Melliar-Smith, and P. Narasimhan. Consistent object replication in the eternal system. *Theor. Pract. Object Syst.*, 4(2):81–92, 1998.

[11] Y. J. Ren, D. E. Bakken, T. Courtney, M. Cukier, D. A. Karr, P. Rubel, C. Sabnis, W. H. Sanders, R. E. Schantz, and M. Seri. Aqua: An adaptive architecture that provides dependable distributed objects. *IEEE Trans. Comput.*, 52(1):31–50, 2003.

[12] A. Z. Spector. Performing remote operations efficiently on a local computer network. *Commun. ACM*, 25(4):246–260, 1982.

[13] E. S. Viddal. Ratatoskr: Wide-area actuator rpc over gridstat with timeliness, redundancy, and safety. Master's thesis, Washington State University, Pullman, Washington, USA, December 2007.