Towards Dependable Grid and Web Services

Geoffrey Fox[1], Shrideep Pallickara[1], Marlon Pierce[1] and David Walker[2]
(gcf,spallick,marpierc@indiana.edu) Community Grid Lab, Indiana University, USA. [1]
Department of Computer Science, Cardiff University, UK. [2]

1 Grid and Web Services

Remote method invocations have been used in distributed systems for quite some time. Frameworks such as CORBA from the Object Management Group (OMG) [1] have had schemes in place to facilitate invocations on remote objects for more than a decade. There also has been support for remote invocations in programming languages, a case in point being the Java Remote Method Invocation (RMI) Framework [2]. In these cases we could think of the remote object as providing a service comprising a set of functions. The provider exposes the service's capability through an appropriate description language, which comprises the function names, the number and type arguments that a given service function takes and finally the return type that would be returned upon completion of the invocation.

The underlying principle for Grid Services [3] and Web Services [4] is similar to what existed in these earlier systems. The difference lies in the scale, scope, ubiquity and easy of utilization of these services. The deployments and utilization of these services are driven by a slew of XML based specifications that pertain to exposing services, discovering them and accessing these securely once the requestor is authenticated and authorized.

2 What's missing?

Service reliability is an important problem for simple point-to-point services. The problem is even more important and difficult in the case of composed services, where a service could comprise interactions involving two or more services. However when a service is composed of multiple services it needs to manage and deal with the unpredictability concomitant in every one of the constituent services. Furthermore, a given service may be accessed and utilized by entities with myriad device profiles.

Increasingly, there is a need for services to adapt to the environments in which it operates. However a service designer should not have to burden oneself regarding the capabilities of the accessing devices or the nature of the networks in which the invoking entities operate in. An entity should also be able to discover services easily, while providing authentication and authorization information that these services might need prior to service invocation (utilization).

It is also implicitly assumed that the invoking entity has to await a response to an invocation. Depending on the complexity of the operation and the load at the service, response times may vary from a few milliseconds to several hours. It is also possible that failures may occur either at the invoking entity or in transit to the entity. Moreover an entity may choose to disconnect prior to the receipt of a response. If the service designer were to try and support this feature across all the constituent services, the composed service's complexity would increase substantially.

3 A motivating example

We outline some of the issues discussed above in an example below. The example is meant to motivate the complexity underlying even simple operations when taken as a whole. Here we take the example of accesses pertaining to the generation and dissemination of meteorological information. Weather sensors at several locations generate voluminous information, which is then routed to the data centers. The sensor data generally amounts to several terabytes of data. Automated services running at these sites generate synopses from these sensor data. A meteorologist then accesses a service running at the data center to retrieve a synopsis that he/she might be interested in. Upon receipt of this synopsis, usually a large image, the meteorologist is interested in closely examining special slices of the sensor data from which the synopses was generated. The meteorologist invokes another service at the relevant distribution center to retrieve these slices.

Depending upon the analysis that needs to be performed on this sensor slice, the meteorologist may need to discover services that can not only perform these analyses but are also willing to lend the CPU time for this possibly compute intensive operation. Once such a service has been located the meteorologist invokes this service to process the spliced sensor data.

The result from this operation is then routed back to the meteorologist who may wish to share the results and the new forecast with other meteorologists to confirm the prognosis. The meteorologist's weather center then proceeds to update its forecast. Meanwhile, a graduate student develops a popular portal which displays weather forecasts by accessing services hosted at the weather center. It is also possible that another user many access this service a PDA device.

4 What are the things that could go wrong

Each interaction within the scenario presented above brings up a set of problems that needs to be addressed. Voluminous data from the sensors need to be delivered reliably to the data centers. The data centers need to route the generated synopses information to all Weather Centers interested in receiving a given synopsis. A meteorologist needs to be able to dynamically locate services for special processing. The meteorologist must be able to retrieve the results from the specialized processing on sensor data slices even after a prolonged disconnect. Collaboration must be fluid and it should be possible to queue requests to collaborators who may not be physically logged in at a given instant.

Some services within the system will be accessed more heavily than others, while other may entail a lot of compute intensive operations. It should be possible to scale certain services by making them highly available and ensuring that accesses to these replicated services is efficiently load-balanced in real time. Finally, it should be possible to make several of these services accessible to pervasive devices. This would entail schemes to reduce network traffic and deployment of specialized error correction routines.

The interactions which various entities (users and applications) and services have with, and among, each other needs to take place in the presence of failures, pending recoveries after failures and prolonged entity disconnects. Furthermore, as the scale and the complexity of the interactions within these systems increase the strains imposed on a service to manage its interaction with other services increases considerably. Ultimately, a service may need to maintain active concurrent connections with all the services that it needs to interact with.

The services itself may be distributed. However, if composed services continue to interact with each other in a point-to-point fashion the strains imposed on resources (network, memory and CPU cycles) would be severe. What this results in, is a transition towards building strongly coupled services the constituents of which were originally intended to be loosely coupled. Each service is responsible for satisfying Quality-of-Service (QoS) constraints with each of the services that it interacts with, each corresponding to an endpoint with whom interactions need to be managed.

Complexity of these services would increase especially when interactions need to be managed in the presence of failures, recoveries and prolonged disconnects. The problems are exacerbated when services need to manage complexities associated with the discovery of services while ensuring secure accesses and exploiting availability of the hosted services.

5 Autonomic Substrates

There are three specific aspects of fault tolerance that are relevant within the context of entities and services within the system. We may enumerate these below –

(a) Aspects which are specific to an entity or a service.
(b) Aspects which are generic to entities and services.
(c) Aspects which pertain to inter-service communications and service-entity communications.

We are proposing an *autonomic substrate* that addresses item (c), much of item (b) while providing support for item (a). The autonomic substrate provides a service for managing and coordinating accesses involving entities and services within the system. The autonomic substrate could be thought of as a malleable substrate that permeates the system, with possibly millions of service points each equivalent to the other. The appellation for the substrate signifies the three core properties that the substrate should satisfy – robustness, reliability and resiliency. Robust corresponds to its construction with minimal assumptions regarding the realms in which it would operate. Reliable corresponds to the ability of the substrate to provide consistent, predictable, correct and dependable behavior. Resiliency corresponds to the ability of the substrate to recover quickly and accurately from failures.

The autonomic substrate is based on building a robust, reliable and resilient distributed messaging infrastructure and facilitating the efficient dissemination of messages within the system. These messages encapsulate information pertaining to transactions, data interchange, system conditions and finally the search, discovery and subsequent sharing of resources. Messages also contain policy information associated with them. The policy information encapsulates reliability, retransmission, ordering, causality, security, compression and other related information. It is the messaging infrastructure's responsibility to ensure that the policy requirements within these messages are satisfied.

The smallest unit of this infrastructure that would provide a back bone for routing interactions between entities needs to be able to intelligently process and route messages while working with multiple underlying network communication protocols. We refer to this unit as a *broker* where we avoid the use of the term *server*s to distinguish it clearly from the application servers that would be among the sources/sinks to interactions generated within the system. The autonomic substrate performs its function through the exchange of messages between brokers, entities and services. It can thus be also looked upon as a distributed messaging infrastructure.

As far we are concerned for the most part the messaging infrastructure and the autonomic substrate that builds upon it as indistinguishable from each other. We now proceed to enumerate the functionalities that need to be provided by the autonomic substrate.

1. Scaling: The underlying distributed broker network should scale to support the increase in managed entities. Inefficient organizations can lead to a broker network that can induce large communication overheads while also being susceptible to partitions. This calls for efficient organization of the broker network to ensure that the aforementioned degradations do not take place.
2. Communications: The transports protocols deployed for communications need to exploit the unique characteristics of the realms in which the substrate is deployed. This implies that the substrate needs to support a wide variety of communication protocols. Furthermore, it is inevitable that the realms we try to communicate across would be protected by firewalls stopping our elegant application channels dead in their tracks. The messaging infrastructure should thus also be able to communicate through firewalls and NAT boundaries.
3. Failures: There should be no single point of failure within the system. It is entirely conceivable that the substrate brokers might fail and remain failed forever.
4. Manage service advertisements: The substrate should manage service advertisements and organize them efficiently. This is a precursor to the discovery process that efficiently and rapidly locates these services.
5. Discovery of Services: Services should be able to advertise themselves. But it is also required that Information Services provide correct information. A problem with the Universal Description, Discovery and Integration (UDDI) registry of services is that the information is generally out of date. The OGSA (Open Grid Services Architecture) **[5]** has a minimal solution: soft state leases that clean up after a while. But there are clearly situations when one may need or want to go beyond that. It should be possible to efficiently discovery services while also specifying constraints and QoS constraints that these discovered services should provide. The discovery of services should facilitate specification of standard querying techniques such as XPath, XQuery and Regular expressions.
6. Management of services: This deals with ensuring the management of service lifecycles and issues related to service versioning. Finally, the substrate should also deal with service migrations.
7. Replication Management: The substrate should facilitate the replication of services and manage access to these services in a way which exploits the high availability provided by the aforementioned replications.
8. End to End Security: The substrate should facilitate secure interactions between communicating entities. Since it is entirely conceivable that messages (including queries, invocations and responses) would have to traverse over hops where the underlying communication mechanisms are not necessarily secure, a security infrastructure that relies on message level security needs to be in place.
9. Robust messaging: The substrate should provide robust messaging for interacting entities. The messaging framework should ensure delivery in the presence of failure, recoveries and prolonged disconnects. The robust messaging should also manage causal (and source) ordering relationships between messages/interaction issued by entities

10. An invocation framework: Once services have been discovered, there should be a mechanism in place to ensure that the service invocation can be carried over the broker network to the service. The exact location of these services might be transparent to the invoking entity.
11. Monitoring Services: State of the substrate network fabric provides a very good indicator of the state of the managed services/entities in a given realm. Monitoring the network performance of the connections originating from individual brokers enables us to identify bottlenecks and performance problems, if any, which exist within sections of the substrate. This information would then be used to project, predict and circumvent failures. Such a scheme also facilitates monitoring the state of services in terms of produced/consumed messages.
12. Notifications: The substrate should incorporate notification services which issue alerts while performing error and diagnostic reporting upon detection of failures, recovering services and service management related notifications. The notification service should also facilitate the instantiation of replicas to circumvent existing/future failures in services.

Also, it is worth noting that service implementation reliability at some level is out of scope of the substrate. Service implementations will have a varying amount of robustness/reliability built in: parameter checking, internal performance and load balancing, etc. The substrate does not make any assumptions about how well a service is written. However, it does detect when something is wrong and notifies the entities interested in utilizing that service.

6 What are the advantages?

There are several advantages to this approach of delegating complexity management to the autonomic substrate. Entities interact with each other through a single access point into the system. There could conceivably be millions of these access points for the substrate, thus providing a great degree of resilience to failures.

Services can acquire reliability and robustness simply by their placement within the substrate. The autonomic substrate infuses these properties into the service without a need for the service implementers to change either the interfaces (service advertisements) or their underlying implementations.

Services clearly benefit from the elimination of networking complexity. It is the substrate's responsibility to permeate through firewalls, NAT, proxy and VPN boundaries. Moreover since the substrate deploys transports efficiently the service's resources are freed to deal only with the function provided instead of managing networking and computational overheads related to communications.

Services also do not need to manage the complexities involved in robust messaging with multiple services. Managing the robust messaging related complexities within individual services would make even the simplest of applications, composed of a small set of services, complex. The problems are compounded when one needs to deal with entity/service disconnects.

Since management of service versioning, replication and load balancing is now managed by the substrate, entities benefit from the high availability of services and invocations on the correct service instances. Service instances, on the other hand, are not overwhelmed by requests from entities since the invocations are now load-balanced by the substrate across the replicated service instances.

Individual services also are able to interact with a wide variety of devices efficiently, since it delegates the management of communications to the substrate. This management would involve content adaptation, compression of data and deployment of specialized communication protocols.

Finally, discovery of services would be efficient without the need to funnel every discovery request through a central clearinghouse. Furthermore, the substrate with its ability to federate multiple search mechanisms (which would also include P2P search mechanisms over the edge) is bound to produce a richer set of results to choose from.

Revisiting our earlier example, the sensors now delegate the robust delivery of the sensor data to the RRR substrate. Similarly, it is the substrate's responsibility to ensure that the weather synopses are routed efficiently to meteorologists (some of whom may retrieve this information after a failure/disconnect). A meteorologist might use the same access point to discover services and to authenticate and utilize the specialized service provided. The meteorologist can retrieve these results after a planned/unplanned disconnect or after a failure from some other device, geographical location and service access point. The underlying substrate has to manage the robust delivery of results to the correct location. The meteorologist can also seamlessly collaborate with other meteorologists in a secure and robust fashion.

Depending upon the strains imposed on the service provided by the weather center, the substrate can manage the replication and invocation of services in a manner, which ensures that strains imposed on individual replicates are minimized, while the entities requesting these services experience good and consistently predictable response times.

Finally services need not provide any specialized pre-processing based on a requesting entity's device/network profile. The substrate manages the complexity of operating in unpredictable and lossy network environments while dealing with the preprocessing (based on the entity's device profile) of messages for delivery to the entities.

7 Conclusions

In this article we discussed the problems that will likely arise as the number and complexity of services increases along with the interactions that exist between these services. Our proposed solution addresses most of these problems. This is a large area of research the scope of which may have far reaching implications. This is a complex problem the complete solution to which may take a significant amount of time. The underpinnings to this substrate and some of the core functionalities are being added to the messaging infrastructure NaradaBrokering **[6,7]**, which is an open source project at the Community Grid Lab at Indiana University.

8 References

[1] The Object Management Group http://www.omg.org

[2] The Java Remote Method Invocation Framework http://java.sun.com/products/jdk/rmi/

[3] The Open Grid Services Architecture http://www.globus.org/ogsa/

[4] The W3C Web Services Acivity. http://www.w3.org/2002/ws/

[5] The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. I. Foster, C. Kesselman, J. Nick, S. Tuecke, Open Grid Service Infrastructure WG, Global Grid Forum, June 22, 2002.

[6] The NaradaBrokering Project at IU Grid Computing Laboratory. http://www.naradabrokering.org

[7] Shrideep Pallickara and Geoffrey Fox. NaradaBrokering: A Middleware Framework and Architecture for Enabling Durable Peer-to-Peer Grids. Proceedings of ACM/IFIP/USENIX International Middleware Conference Middleware-2003.