1.1 Gridable Service

One unique aspect of SOA vs. other architecture styles like Object Orientation , Client/Server or even 3-Tier architecture is that it is built for highly distributed systems. Each and every service is a sub-system in itself it can run on its own machine and be located everywhere in the world . Many times, the service itself needs to be distributed in its own right. One reason to use distributed computing inside the service is computational intensive tasks.

One of my recent projects was the development of a biometric platform. The platform can be used for many usage scenarios. A simple scenario is an access control systems - e.g. authorize entrance into a secure building or area. This is a relatively simple scenario as you usually only have to deal with few thousands of people and as a person requests entry she also declares who she is (e.g. using an RFID card with her ID). In these cases you can go to the database, lookup the appropriate record , run the biometric algorithm or algorithms and verify the person is who she says she is. However the same platform also has to work for other, much more demanding and computing intensive scenarios. For example consider a forensics scenario where you have a fingerprint collected at a crime scene, in this case you don't know who the person you are looking for is, and you have to run your search on basically all the database which can contain millions of records. Keep in mind that when you match a biometric template[1] you calculate the probability of a match (based on the internal structure of the template) and that each template weights about a one kilobyte you quickly realize that this can be quite a CPU intensive task.

Sometimes when you develop you SOAs you will have algorithmic tasks or other computational heavy tasks such as the one mentioned above and the question is

How can a Service handle computational heavy tasks in a scalable manner?

Robert Anderson and Daniel Ciruli covered some of the problems of trying to handle computational heavy tasks (Anderson & Daniel, 2006). One option is to try to scale up, just throwing bigger iron at the problem will work to an extent, until you reach the limitation of the machine. The other problem with this approach, which is more severe, is that this type of solution gets real expensive really fast. Especially if you also have availability requirements which means you'd have to have at least two machines that are capable of handling the computational task

Another approach is to try to scale-out instead of scale up. Scaling out will ultimately makes the solution less expansive (compared with a single huge server), but you still need a way to distribute the work between the multiple nodes. One way to do that would be to use network load balancing. This would work for problems of having a lot of requests to do simple tasks. However it won't be very good in computational intensive tasks. Also, while it might be possible to take a problem that can be solved by parallel processing and hand it to a NLB network, you would need to add a lot of layers on that (like management, make sure the tasks are small enough so NLB scheduling algorithms like round robin etc. would not just pile work on the nodes, tracking

[1] you can think of biometric template like a signature or a hash that represents the biometric sample. The template is smaller than the sample but contains enough information to identify the original.

progress etc.). Thus while the scaling out looks promising price-wise it seems we need to stay with option A after all – unless we

Apply the Gridable Service pattern and introduce grid technology inside the service to handle computational heavy tasks

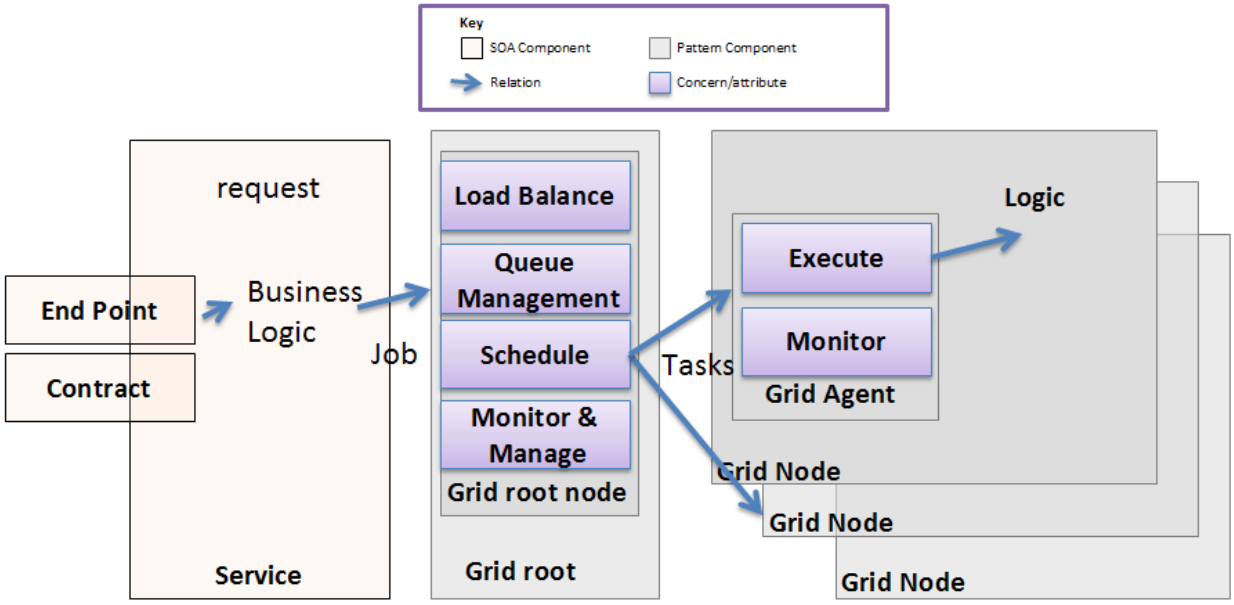

Figure 2.x Gridable Service Pattern

The Gridable Service pattern is based on deploying a computation (and possibly a data) grid as part of the internal structure of a service. When the service business logic needs to handle a chore that is computation heavy (like running some financial analysis, fusion algorithms etc.) the business logic creates a job on the grid root. The grid then queues the request. A Job is made of one or more tasks that can be executed on the grid. The scheduler distribute the tasks to one or more nodes (depending on the job type sequential, workflow, parallel) where they are executed by the grid agent.

The Grid infrastructure. (Agent, root node etc.) constantly monitors available resource. Adding a new hardware (which has the agents and all the software installed) enlarges the pool of available resources. The grid takes care to maximize the usage and does that based on the load on the machines (not just the amount of requests / round robbing as NLB does). The smart resource allocation helps solve both scalability and balancing scenarios. The grid also helps with availability scenarios by redirecting jobs when one of the nodes fails. The Gridable service pattern can be combined with the Workflodize pattern e.g. by making the tasks in the job workflow instances or by making a workflow drive the jobs

Future trend: Software as a Service

Grids have another aspect besides performance - one pattern (actually a set of patterns) that is starting to emerge that can rely on grid technology, is Software as a Service (SaaS). SaaS

overloads the term Service (yet again) and in essence it is a way to get utility computing (e.g. the ability to treat your computing needs the same way you treat electricity). SaaS goes well beyond the mere hosting of your services by a third-party and consuming them over the internet as it introduces challenges in integration (security, transformation etc.), assuring SLAs, data security and the whole IT roles and responsibilities in the organization. . In a sense, a successful SaaS is a threat to the IT organization (as we know them today) as it can help realize Nicholas G. Carr's vision of "IT doesn't Matter" (Carr, 2003) and commoditize IT, at least to an extent. This is definitely an architectural approach and a set of technologies to keep on your radar screen

Let's go back to the biometric problem presented earlier to better understand how all that works. One of the services defined in this system is a Pattern Matching service which takes a biometric pattern (sort of a hash for a biometric sample) and tries to find matches in the patterns database. This is a time consuming effort as database sized can get to hundreds of millions of templates (per engine) on top of that you need to use the biometric engine to compare the templates as some bits are more important than others (e.g. the distance between the eyes is more important than a beard for a face recognition scenario). Figure 2.X shows how the problem can be solved using the Gridable Service pattern.

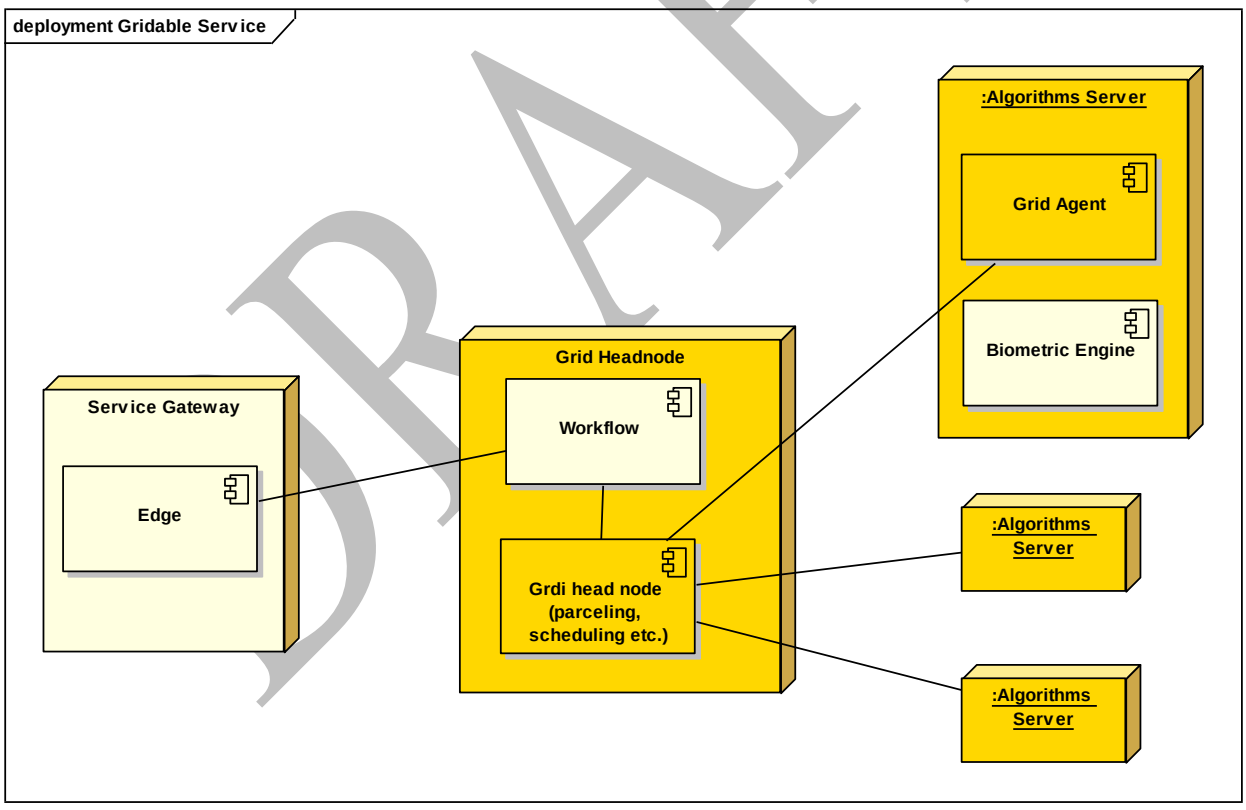

Figure 2.x Gridable Service pattern applied to one of the services of a biometric platform. The different biometric engines (finger, face recognition etc.) are deployed on the grid and a workflow drives their invocation (depending on the requests it gets).

The edge component translate the request to the internal representation and invokes the workflow that deals with matching. The workflow component then works with grid head node to

partition the matching job and schedule it. The grid infrastructure will take care of finding free Algorithm servers and invoking the biometric matching engines there.

Let's look at some of the options for implementing this and similar scenarios.

1.1.1 Technology Mapping

There are many Grid implementations – all of them can be applied in an SOA context to implement the Gridable Service pattern.

One notable effort in grid computing is the Gridbus project which tries to create open-source specifications, architecture and a reference Grid toolkit implementation for Service Oriented Grid (and utility computing which I'll discuss as part of the Orchestrated Choreography pattern).

In Grid scenarios you just create remote threads of execution - without knowing where the execution will take place. The grid infrastructure optimizes the available CPU cycles , hardware etc. on all the connected nodes and execute the job/thread on a free machine. Diagram 2.X below shows a screenshot of the system console for Alchemi.Net which is a Microsoft .Net implementation of the Gridbus standards.

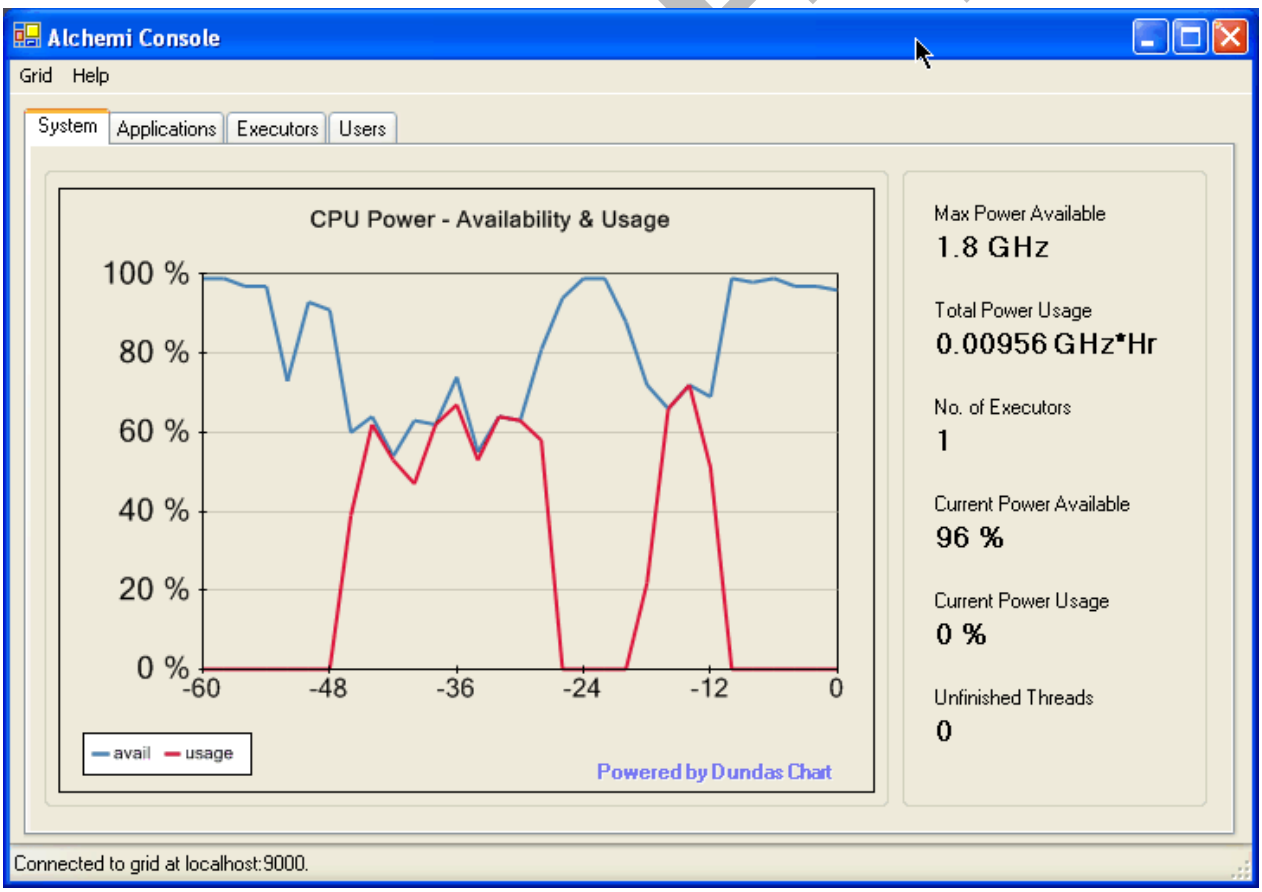

Figure 2.x Management Console of Alchemi.NET a Microsoft .Net based open source project implementing GridBus

Gridbus is, of course, not the only game in town. In the Microsoft world you have the relatively new HPC Cluster (where HPC stands for High Performance Computing). And there

other vendors like Digipede etc. On the Java front also have quite a few implementations like CSF (Community Scheduler Framework, GridWay and many others.

Lastly, the WS* protocol stack also takes care of grid scenarios and there are few protocols bundled under the name WS-Resource Framework (WSRF). WSRF actually includes 5 protocols:

- WS-Resource – defines the relation of a resource on the grid to web-services
- WS-ResourceProperties – a protocol to retrieve and set the list of features/properties of each resource
- WS-ResourceLifetime - The semantics to control the life time of a resource (i.e. create and destroy resources)
- WS-ServiceGroup – a standard for defining a collection of resources
- WS-BaseFaults – handle problems and faults

The nice thing about all grid technologies is that they help elevate some of the concerns that otherwise would take a lot of your time to implement.

1.1.2 Quality Attribute Scenarios

Gridable Service (and the grid technology it builds upon) can help with some of the common quality attributes projects face (like performance or availability).

All the quality attributes are met using the same mechanisms that allow redistribution of computation loads based on the available resources, e.g. scalability is addressed by the fact that resources are pooled and constantly monitored. The gird is able to reroute work in case of failure as well a redistribute the load when a node it added.

Here are a couple of examples for scenarios that can point you to looking at the Active Service pattern.

Quality Attribute (level1)	**Quality Attribute (level2)**	**Sample Scenario**
Performance	latency	Under normal conditions analyzing a customer profile (running the static analysis algorithm,) with will take no more than 30 seconds
Availability	Hardware failure	Upon a server crash the system will resume operations in 30 seconds (allowable performance degradation TBD)
Scalability	Scale-out	Under all conditions, it would be possible to deal with higher order loads by adding more hardware
Budget	Hardware costs	When doing capacity planning strive to have the least possible number of servers

One important quality attribute that is missing here is security. This is not a core capability of the grid as a concept. Note however, that the serious grid implementations should have a solid security solution

Gridable Service can help you solve some of the basic needs of distributed systems (performance and availability). The grid can also serve as a scalability solution. However, Grids are not the only way to achieve scalability. Take a look for a example at the Service Instance pattern