Mapping Engineering Design Processes onto a Service-grid: Turbine Design Optimization

Sanjay Goel, Shashishekara S. Talya, Michael Sobolewski

CE CONCURRENT ENGINEERING: Research and Applications

Mapping Engineering Design Processes onto a Service-grid: Turbine Design Optimization

Sanjay Goel,[1,*] Shashishekara S. Talya[2] and Michael Sobolewski[3]

[1]*School of Business, University at Albany, State University of New York, BA 310b 1400 Washington Avenue, Albany, NY 12222*
[2]*Product Engineering GE Energy, Gasification, Houston, TX*
[3]*Department of Computer Science, Texas Tech University, Box 43104 Boston and 8 th St., Lubbock, TX 79409*

Abstract: This study presents an application of a distributed service-oriented architecture (FIPER) for the preliminary design of gas turbines. The FIPER architecture is based on the concept of registration and discovery of services in real time. It uses a service catalog that registers the services started on the network. These services are then discovered in real time from the grid and used in the design process. The turbine preliminary design process involves changing the configuration of the turbine incrementally and evaluating its performance using analysis (simulation) code. During the design process a one-dimensional analysis code is wrapped into a service using a standard interface developed in FIPER and launched on the network. Several such services were distributed across a grid of workstations. These services were then used to support the turbine configuration optimization process. The study presents the results of the optimization as well as the scalability characteristics of the service-grid.

Key Words: service-oriented computing, grid computing, turbine design, business process reengineering, FIPER, engineering automation, network resilience.

1. Introduction

Design of engineering systems, such as gas turbines, involves a complex process with multiple steps and several feedback loops. The design entails an interaction between several engineering disciplines, such as heat transfer, aerodynamics, and structural analysis. The objective of the design is to satisfy the problem constraints and to maximize the performance of the design. Performance metrics vary for different types of design, for instance, in a power-generation turbine the objective is to maximize the fuel efficiency and in a fighter-engine turbine the objective would be to get maximum thrust. Such designs do not have a closed form solution; however, analysis codes that can numerically compute the performance of specific design configurations are available. The design thus involves evaluating alternate configurations of the design, from which the design that maximizes the performance of the turbine is selected. Designers, during the manual process, use expert judgments based upon prior experience to select alternate configurations of the design and systematically narrow down the search space. Nevertheless, this is a labor-intensive activity that requires expertise of skilled designers that are not easily available.

Therefore, there is a strong incentive to automate these processes using automation/integration environments that can capture the designer intelligence. The design problem can be posed as an optimization problem and an optimization algorithm can be used to select the best design among the alternate configurations. Such automation has the advantage of faster execution speed; however, its performance in decision-making is poorer compared to skilled designers. The automated design processes thus require a relatively larger number of iterations to determine the solution. The optimization is typically executed on a single node where the design iterations are sequentially run. Although each analysis executes relatively quickly, a computational bottleneck is created, as a large number of analysis runs are required especially when using stochastic optimization techniques, such as genetic algorithms for driving the optimization.

Use of distributed computing eases the computational bottleneck by simultaneously evaluating multiple configurations across a network of services. Use of distributed environments is typically plagued with high maintenance cost and high unpredictability, primarily due to lack of network reliability. However, use of a service-based distributed architecture significantly reduces the maintenance overhead due to its inherent resilience by virtue

*Author to whom correspondence should be addressed.
E-mail: goel@albany.edu
Figures 1, 4 and 5 appear in color online: http://cer.sagepub.com

1063-293X/08/02 0139–9 $10.00/0 DOI: 10.1177/1063293X08092487

of the protocol that allows real-time discovery of services. Using this architecture, complex processes can be automated by mapping multiple services from the network grid into the design process. Changes to individual services can be accommodated easily by adding new services and phasing out old services without any changes to the basic architecture.

This study presents the application of FIPER, a service-based architecture developed for distributed computing in an engineering environment, to solve the turbine design problem. The work presented in this study utilizes the architecture developed at the General Electric Company under the National Institute of Standards and Technology Advanced Technology Program (NIST-ATP) Federated Intelligent Product EnviRonment (FIPER)[1] contract [1]. The goal of the FIPER program was to develop technology to reduce design cycle time, and time-to-market by intelligently automating elements of large scaled distributed design processes in a linked, associative environment. It is important to note that the goal of the FIPER program was to develop and demonstrate technology, which differs from the commercial FIPER software product developed and marketed by ESI [2]. Any reference to FIPER in this manuscript is to the work done under the FIPER contract and is not directly related to the commercial software product.

Turbine design problems start with a preliminary design that uses 1-D analysis and then progresses to 2-D and 3-D design. In the design process the computational complexity of the analysis increases as the design advances along the process. The automation of the design process is more useful when the computational effort is relatively much lower compared to the manual effort. Since designers are typically more efficient in narrowing down the search space compared to search algorithms, the payoff in automating the design process is much higher when using analysis codes with lower computational burden. In the turbine design problem, the analysis codes are least computationally intensive during the preliminary design. The study focuses on the problem of preliminary design of the turbines and presents the optimization formulation as well as the application of the FIPER architecture to the problem.

The rest of the study is organized as follows: Section 2 presents a review of the relevant literature. Section 3 presents the FIPER architecture, the turbine design problem formulation and the application of the FIPER architecture to the problem. Section 4 presents the performance results of the distributed computing model. This is followed by some concluding remarks and plans for future work.

2. Literature Review

Two streams of literature are relevant to this research, namely, design process automation and distributed analysis environments. Traditionally, design process automation tools are used for automation of processes on individual nodes of the network. Resources on the web are occasionally used via use of remote execution scripts that distribute the computational burden on the grid. While simple to implement, these scripts are brittle with little error correction and resilience built in. Distributed analysis environments on the other hand are robust in remote execution of analysis codes on the network but lack the basic infrastructure for process automation. Executing of engineering processes on a grid of workstations requires both a distributed execution environment and automation tools. The literature review discusses the two streams of work in the context of the FIPER distributed automation/execution environment.

Design process automation requires mapping the design tasks into a process map which may include loops, branching, and transitions. The tasks typically involve analyzing different designs for performance, structural integrity, resilience, reliability, etc. The design usually does not have a closed form solution and requires repeated execution of an analysis code in which alternate configurations of the design are evaluated. Automation of these tasks requires optimization models that drive the design to maximize (or minimize) a set of objectives subject to the constraints of the problem. The entire design process is then simulated by executing the codes in sequences provided in the process map and cascading outputs from one analysis code to the next. Several attempts have been made in the past to create a collaborative design environment through the automation of tasks in the design process, [3–6] with an objective of design cycle time reduction and performance improvement. These automation efforts, though robust at the individual task level where the process is standardized [1], are brittle at the process level. Brittleness here refers to inflexibility and inability to adapt to changes. The reason for brittleness at the process level is that analysis codes, as well as the process, change and render the couplings between tasks ineffective thereby breaking the process map. The maintenance burden of fixing broken links is onerous and makes the automated processes inefficient in terms of the maintenance cost versus productivity gains.

In discussing distributed analysis environments P2P systems need to be distinguished from distributed computing. Distributed computing involves breaking down a computationally large task into several subtasks that are distributed over the network, while P2P systems engage in direct communication among nodes. While some of the

[1]FIPER contract was a four year (1999–2003), $21.5 million NIST-ATP project that teamed General Electric with Engineous Software Incorporated (ESI), Goodrich, Parker Hannifin, Ohio Aerospace Institute, and Ohio University.

issues in the two domains overlap, this discussion is primarily focused on distributed computing. The problem of coordinating distributing computation across multiple network nodes has also been investigated under the rubric of parallel computing[2] and meta-computing (grid) systems. Grid computing organizes computational and data resources distributed across a network to make computationally intensive problems feasible to solve. Most of the distributed computing systems are based on centralized coordination; however, some distributed computing systems based on P2P architecture are beginning to emerge. One notable application is Seti@home [7], which shares processing load across distributed nodes. The application divides processing into small chunks, distributes them to other nodes, and then reassembles the results to obtain the overall solution. The application achieves resilience through redundancy, i.e., by using multiple nodes to perform the same computation task and eliminating specious or incongruous results. I-Way [8] is another project that demonstrates the feasibility of sharing distributed resources. In the I-Way project, multiple super computers process execution tasks and communicate over high-bandwidth ATM networks reducing execution time for complex analysis.

Grid computing poses several challenges, including scheduling, coordination of activities, access control, load balancing, fault tolerance, and integration of heterogeneous systems [9]. Researchers are just beginning to explore these issues as grid computing becomes more prominent. Interestingly, grid-computing applications employ centralized scheduling architectures. Generally, a central scheduler manages the distribution of processing tasks to network resources and aggregates the processing results. These applications assume a tightly coupled network topology, which does not automatically adapt to the changing network topology. Load sharing and job scheduling schemes have been studied extensively with formal performance evaluation models [10,11]. Generic grid computing toolkits that can be adapted to multiple applications, include, Globus [12], Legion [13], Simgrid [14], and Globe [15]. These toolkits provide support for basic elements of distributed computing, such as communication, resource location, scheduling, authentication, and security through a centralized control. Other toolkits have been developed for specific applications, such as GradSolve [16], Ninf [6], and NetSolve [17], and are based on remote procedure calls.

The FIPER design environment bridges the gap between distributed environments and process automation tools, creating a comprehensive tool that allows complex engineering processes to be mapped to a network grid. It breaks the tasks into services on the network that are coordinated via a central control. This architecture has inherent resilience by allowing services to be discovered in real time such that a disabled service can be replaced by locating another service in the network without disrupting the process. The architectural details are discussed in the next section.

3. FIPER Architecture

FIPER is a service-to-service (S2S) distributed environment based on the Jini Network Technology [18–20], which supports a federation of services that collaborate dynamically over a network. In Jini, a service is essentially a Java interface that is implemented as a remote object. Therefore, any object implementing multiple interfaces could be turned into a provider of multiple services. The Jini service-oriented architecture has a concept of dynamic discovery and join of services whereby services are registered on the network and discovered in real-time via a unicast or multicast protocols on the network.

Jini provides a registry called lookup service (LUS), which is a service registry that allows service requestors to locate needed services by object types (interfaces) and associated complementary attributes. During startup, a service provider registers its services with the LUS. Clients use the LUS to locate the services they are interested in. The LUS itself is discovered through the discovery protocols by issuing multicast or unicast requests, as well as by receiving multicast announcements. Service requestors and providers use the discovery protocols to locate LUSs. In FIPER, discovery of LUSs is delegated to a specialized service provider, called *cataloger*, which maintains a catalog of domain specific services from all available LUSs. Multiple catalogers are usually maintained on the network to partition all services into dynamic application specific groups across all running LUSs (aero, thermal, mechanical, analysis, etc.). When the services first enter the grid they receive a lease from a LUS for a specific time period which is renewed periodically by their service provider. If the service gets disabled, then the lease is not renewed and the service is deregistered from the LUS and thus leaves the network. This mechanism of leasing keeps the grid healthy and robust. New services entering the network become available immediately via a cataloger or directly from LUSs and the existing services that are disabled are automatically disposed from the grid.

FIPER allows a virtual mapping of an engineering process on a grid of virtual services. This mapping is defined by a service-oriented program, called an *exertion*, in terms of the data for each service, the service operations to be invoked on each associated service

[2]Traditionally, researchers and practitioners have called distributed resource sharing parallel computing; however, since the mid-nineties, grid computing is more commonly used, especially as it relates to high performance distributed computing.

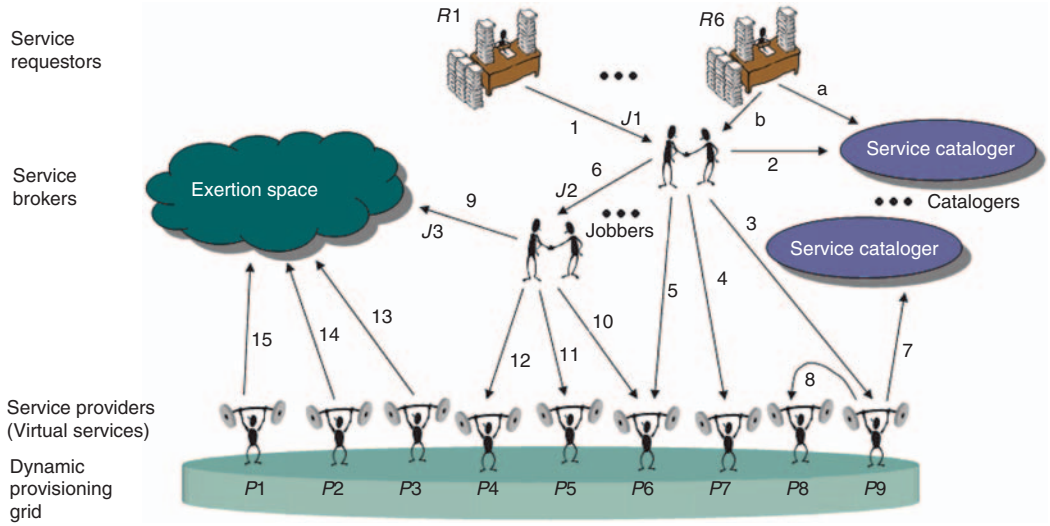

Figure 1. FIPER layered architecture (requestors, brokers, providers, provisioners).

provider, and the control strategy defining which operations use when and where. A service operation is defined by what is called a *service signature*. A service signature is a reference to a remote method implemented by any service provider in the network. Thus, a service provider runs a set of remote methods which are exposed to service requestors via a provider's interfaces. A service signature is a pair (i, o), where i is the name of the provider interface and o is the name of operation (selector). Service data that are passed as an argument to the remote call is called a *service context*. A service context [21,22] is a tree-like structure with leave nodes containing data and tree paths providing the context namespace. A *task* is an elementary grid operation that is defined by data and a service signature, i.e., task $t = (c, s)$, where s is the service signature and c is the service context. A composite exertion is called a *job*. It is an aggregation of tasks and other jobs. The job defines a virtual mapping of engineering process and also encapsulates the control strategy for the job execution. A job $j = (c, s)$ where s is a service signature and c is the service context in which the data nodes are the tasks and other jobs. The context model for a job also encapsulates information required for defining the control strategy of the process represented by the job. Tasks and jobs are in fact grid programs, more precisely grid instructions and procedures respectively and are collectively called exertions.

All the engineering analysis codes are wrapped into services and distributed on the network. A rendezvous service called a *jobber* finds the services that are necessary to complete synchronously the entire process defined by job. A jobber creates and manages a federation of services using the *cataloger* – or finds them in discovered LUSs by itself. FIPER also extends task/job execution abilities through the use of a rendezvous service called *spacer*. The spacer service can drop a task into a shared exertion space provided by JavaSpaces [20] in which several providers can retrieve relevant tasks or jobs from the exertion space, execute them, and return the results back to the object space. While the jobber federation acts synchronously, the spacer federation is asynchronous in nature. These federated services by either jobber or spacer complete the process and disperse to join other federations in the grid.

Alternatively, a service requestor can use an exertion space and simply drop the exertion (task or job) into the shared exertion space. Each FIPER provider can look continuously into the space for exertions that match the provider's interfaces and complementary provider's attributes. The provider that picks up a matched exertion from the object space returns the exertion being asynchronously executed back into the space. Then, the requestor that placed the exertion picks the executed exertion up from the space. The exertion space provides a kind of automatic load balancing—the fastest available service provider gets an exertion from the space. When a service provider gets its task, the task method specified in its signature is executed; otherwise jobs are picked up and executed by jobbers or spacers depending on the exertion's control strategy. To illustrate a very flexible distributed control strategy of FIPER service-oriented programs, let us consider the case presented in Figure 1. This control strategy defines a virtual mapping of an engineering process on a grid of virtual services. A service requestor $R1$ submits a job $J1$ to a jobber (action 1). In that case $R1$ finds a jobber directly by itself (discovery of LUSs and selecting the jobber). In the case of *R6*, the jobber is found using a cataloger (actions *a* and *b*, but the cataloger is found directly by *R6*). The control strategy is driven by the service context associated with $J1$ as it is described above. A service context of each job defines a control strategy by four attributes as follows:

1. Discovery: delegated (cataloger) or self
2. Coordination: delegated (jobber or spacer) or self

3. Dispatch: push (jobber for jobs and matching providers for tasks) or pull (exertion space)
4. Execution strategy: sequential or parallel

In Figure 1, the control strategy of the job $J1$ is defined as follows:

1. discovery method: delegated (cataloger)
2. coordination method: delegated (jobber)
3. dispatch method: push (jobber)
4. execution strategy: parallel

The control strategy of the job $J2$ (the subexertion of $J1$) is defined as:

1. discovery method: self
2. coordination method: delegated (jobber)
3. dispatch method: push (jobber)
4. execution strategy: parallel

Finally, the control strategy of the job $J3$ (the subexertion of $J2$) is defined as follows:

1. discovery method: self
2. coordination method: delegated (spacer)
3. dispatch method: pull (exertion space)
4. execution strategy: parallel

At the inception of the process, the job $J1$ is submitted to a jobber (action 1). The jobber then obtains the proxies to all the required services using the cataloger (action 2). The submitted (outer) job $J1$ by $R1$ is coordinated by the jobber that include actions 3, 4, 5 (executing tasks), and 6 (executing the inner job $J2$ of $J1$). The provider $P9$ calls on $P8$ using the cataloger: actions 7 and 8. The current jobber then finds another jobber (action 6) that coordinates execution of the inner job $J2$. The job $J2$ is executed in parallel – the inner job $J3$ of $J2$ is being executed via actions 9 and 10, 11, 12 – tasks being executed by providers $P6$, $P5$, and $P4$ correspondingly. Job $J3$ is dropped into the exertion space and coordinated by its spacer (action 9) and is executed by providers $P3$, $P2$, and $P1$ in parallel – actions 13, 14, and 15 correspondingly.

Exertion-oriented programming can allow us to perform executions on various service providers, but where does the S2S communication come into play? Often times, a service provider may request the services of another provider to help complete a certain task or job, much like a peer in a P2P system. How do these services communicate with one another if they are all different? Top-level communication between services, or the sending of service requests, is done through the use of a method called *service* (***Servicer.sevice (Exertion):Exertion***) that all FIPER services are required to provide. This top-level service operation takes an exertion as an argument and gives back an exertion as the return value.

So why are exertions used rather than directly calling on a provider's method and passing service contexts? There are two basic answers to this. First, passing exertions helps to aid with the network-centric messaging. A service requestor can send an exertion out onto the network – ***Exertion.exert()*** – and any service provider can pick it up. The provider can then look at the interface and operation requested within the exertion, and if it does not implement the desired interface or provide the desired method, it can continue forwarding it to another service provider who can service it. Second, passing exertions helps with fault detection and recovery. Each exertion has its own completion state associated with it to specify if it has yet to run, has already completed, or has failed. Since full exertions are both passed and returned, the user can view the failed exertion to see what method was being called as well as what was used in the service context input nodes that may have caused the problem. Since exertions provide all the information needed to execute a task including its control strategy, a user would be able to pause a job between tasks, analyze it, and make needed updates. To figure out where to resume a job, the jobber service would simply have to look at the task's completion states and resume the first one that was not completed.

3.1 Turbine Aerodynamic Design Problem

Turbines are complex engineering systems that are composed of multiple alternating rows of stationary and rotating airfoils, which permit controlled expansion of the hot gases from the combustor and generate power in the process. Design of a turbine involves optimizing the thermodynamic parameters of each row of airfoils, shape of individual airfoils, and the system parameters that model dependencies between the rows of airfoils. To ensure its integrity, a turbine must be evaluated from several different perspectives including, thermodynamics, aerodynamics, structural analysis, and rotor dynamics. The turbine aerodynamic design problem does not have a closed form solution; rather, an analysis code can be used to evaluate a specific configuration of the turbine. The design process is thus an iterative process in which several configurations of the turbine are evaluated and the configuration with the best overall performance is selected.

As shown in Figure 2, the design proceeds in several phases such that the computational complexity of the analysis increases progressively as the design advances. Initially a broad range of the search space is examined and it gets progressively narrower as the design advances and the analysis gets more computationally expensive. The design process starts (Cycle) with the thermodynamic cycle analysis of the aircraft engine to determine the flow conditions at the interfaces between the different

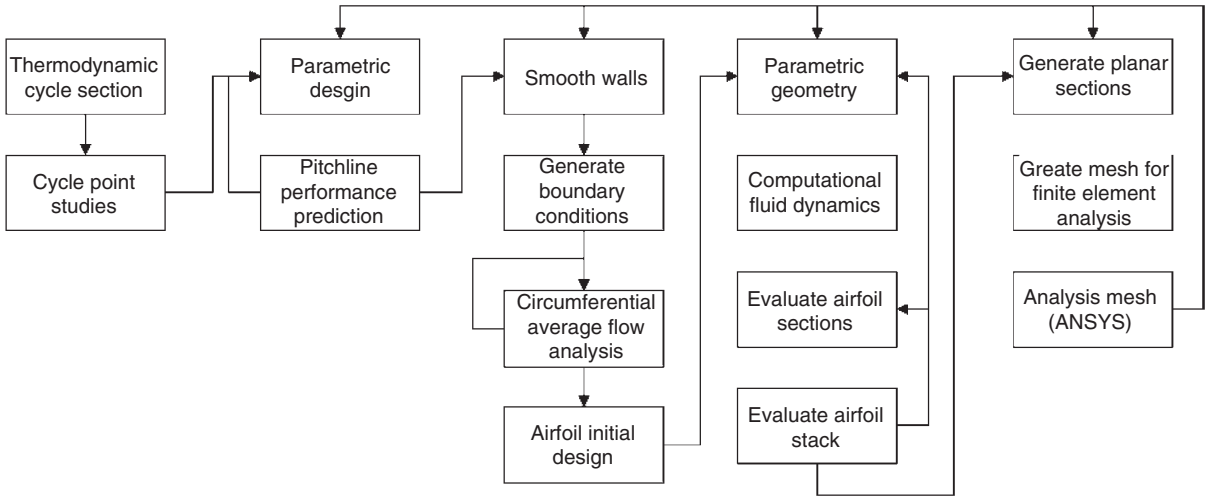

Figure 2. Turbine aerodynamic design process.

components (compressor, combustor, turbine, etc.). Step 0 indicates the 0-D (0-dimensional) fidelity of the analysis. The complete engine is modeled as a thermodynamic cycle and the performance of the engine is evaluated at different points in the flight regime of the aircraft engine. The different flight points, also called as cycle points, can be ground idle, take-off, cruise, landing, etc. The turbine is usually designed for maximum or best performance at a particular cycle point, called the design cycle point and is required to meet the minimum performance criteria at all other cycle points, which are typically referred to as off-design points. Step 1 is the preliminary design in which a one-dimensional (1-D) analysis code based on empirical data is used to analyze all the turbine stages simultaneously in order to predict the turbine performance. This phase plays a crucial role in narrowing down the search space and reducing the computational burden in the downstream phases. During the preliminary design the macro parameters of the turbine, such as the number of airfoils in each row, geometry and configuration of the airfoils, as well as the basic shape of the turbine *flowpath* are determined. In later phases, the focus shifts to the design of individual airfoils where 2-D/3-D computational fluid dynamics codes are used to optimize the airfoil geometry by minimizing friction and leakage losses. The focus of this work is on turbine preliminary design where a 1-D code is used in the analysis.

3.2. Turbine Preliminary Design

Turbine preliminary design involves optimization of the shape of the gas path as well as the configuration of each blade row in the turbine. The turbine consists of rows of alternating blades and vanes, thus, the parameterization of the turbine is typically done based on each stage. The parametric representation of the turbine consists of the blade solidity, reaction, work

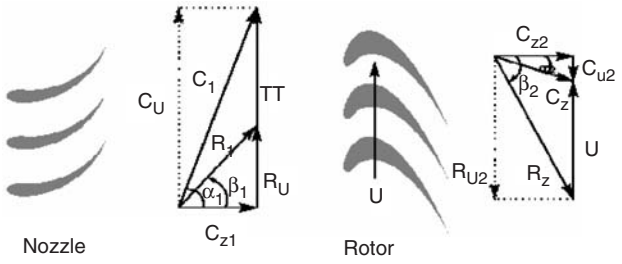

Figure 3. Turbine flowpath calculations.

extraction, inner and outer diameter, and axial width of the blade row. This parametric representation is evaluated using a 1-D analysis code that is based on an empirical model and validated using turbine test data. This code models the performance of each of the blade rows and the interactions between adjacent blade rows. The performance of the turbine is evaluated at multiple thermodynamic cycle points. The goal is to map the performance of the turbine over the entire design space and identify the critical areas of the design wherein the turbine meets all the design criteria and maximizes performance.

Figure 3 shows a typical velocity triangle based calculation that is performed during a 1-D aerodynamic analysis. In this particular application, a GE proprietary code is used to perform the 1-D analysis. The 1-D code is wrapped as a service and whenever it gets published to the network, it is available for service requestors to be able to execute the specific tasks in a particular job. Each service provider is identified by a unique interface and a unique provider name, and the service requestor can select individual service providers based on these attributes. In addition to this, each provider can perform more than one service and depending on the type and number of inputs specified, the provider can intelligently determine the specific service that is requested.

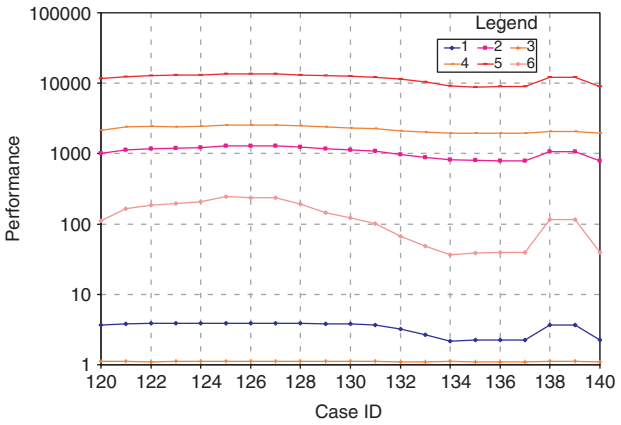

Figure 4. Turbine performance map.

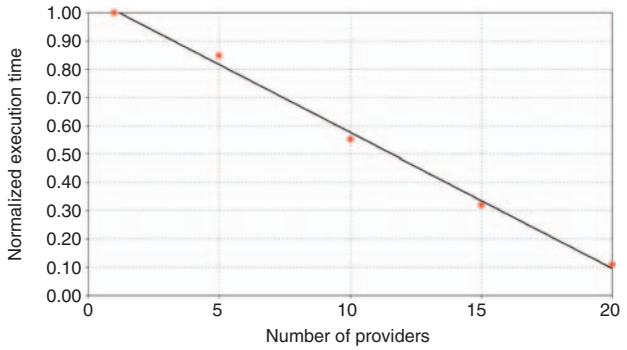

Figure 5. Execution time for different number of service providers.

4. Results

A specific aspect of the turbine aerodynamic preliminary design process is defined as a job and executed using the FIPER framework. The 1-D aerodynamic code is exposed as a FIPER service. The specific example that is chosen requires evaluating the turbine 1-D performance over 20 different thermodynamic cycle points so that the designer can then narrow down the design space to a region that maximizes performance while meeting all the other design constraints. Figure 4 shows the turbine performance map for the 20 thermodynamic cycle points. The Case IDs are plotted on the *X*-axis and the performance parameter is plotted along the *Y*-axis. Actual names of performance parameters have been substituted by numbers to protect the proprietary information at general electric.

To obtain these results, the 1-D analysis code has to be executed 20 times and the output from the analysis has to be post-processed to extract the necessary performance parameter. For a typical turbine design, hundreds of such analyses are executed in order to accurately map the entire design. The process is typically performed sequentially, resulting in a very time consuming process. The strength of a distributed environment is in its ability to perform such operations in a parallel environment. The 20 analyses were defined as 20 individual jobs and were spawned to different number of service providers. The data was collected on the time it takes for all 20 jobs to complete.

Figure 5 shows the data collected when the 20 jobs are all run on 1, 5, 10, 15, and 20 service providers. The actual execution time is plotted as red-colored points and a trend line is also plotted to indicate the trend. In the case where more than one job is sent to a particular service provider, the first job has to execute before the next job can be executed. The execution time for the different cases is normalized with the execution time for the case with one 1-D service provider. One can observe the liner trend in the execution time going from 1 to 20 providers. There is a 90% reduction in the time it takes to execute 20 jobs using a single service provider as compared to using 20 service providers. The slight degradation in performance is caused due to the network traffic at the time of execution and due to network latency during the process of provider lookup and discovery.

5. Conclusions

Use of a distributed service based environment reduces the throughput of the design process via parallel execution of analyses over the network. This study demonstrates the use of this architecture for optimization of a turbine where multiple services are executed in parallel. The process scales almost linearly as more and more services are added to the network. Since the architecture uses a dynamic service discovery mechanism allowing new services to enter the network and disabled services to leave the network with need for reconfiguration. This allows the process to be distributed without sacrificing the robustness of the process. This architecture also improves the utilization of the network resources by distributing the execution load over multiple nodes of the network. The process shows resilience to service failures on the network as alternate services can seamlessly replace disabled services such that continuity of operations is not disrupted. Such architecture can provide robustness to computing environments that face ever increasing threats from viruses, worms, hacking, and failures.

References

1. Federated Intelligent Product EnviRonment (1999). Technical Proposal, Ohio Aerospace Institute, General Electric Company, BFGoodrich, Parker Hannifin, Engineous Software, Ohio University, April 1999.
2. Engineous Software Incorporated. http://www.engineous.com/index.htm

3. Goel, S., Cherry, D. and Gregory, B. (1993). Knowledge-Based System for Preliminary Aerodynamic Design of Aircraft Engine Turbines, *Applications of Artificial Intelligence XI: Knowledge-Based Systems in Aerospace and Industry*, Florida: Orlando.
4. Kolb, M.A. and Bailey, M.W. (1993). FRODO: Constraint-Based Object-Modeling for Preliminary Design, *Advances in Design Automation*, pp. 307–318.
5. Takefusa, A., Matsuoka, S., Ogawa, H., Nakada, H., Takagi, H., Sato, M., Sekiguchi, S. and Nagashima, U. (1997). Multi-client Performance Analysis of High-Performance Global Computing, *Proc. 1997 ACM/IEEE Supercomputing Conference*.
6. Takefusa, A., Matsuoka, S., Ogawa, H., Nakada, H., Takagi, H., Sato, M., Sekiguchi, S. and Nagashima, U. (1997). Multi-client Performance Analysis of High-Performance Global Computing, *Proc. ACM/IEEE Supercomputing Conference*.
7. Anderson, D., Cobb, J., Korpela, E., Lebofsky, M. and Werthimer, D. (2002). *SETI@home: An Experiment in Public-Resource Computing*, U.C. Berkeley: Space Sciences Laboratory.
8. DeFanti, T., Foster, I., Papka, M., Stevens, R. and Kuhfuss, T. (1996). Overview of the I-Way: Wide Area Visual Supercomputing, *International Journal of Supercomputing Applications and High Performance Computing*, **10**(2): 123–131.
9. Johnston, W., Gannon, D. and Nitzberg, B. (1999). Grids as Production Computing Environments: The Engineering Aspects of NASA's Information Power Grid, *Proc. Eighth IEEE International Symposium on High Performance Distributed Computing*.
10. Ingram, D. (1999). Soft Real Time Scheduling for General Purpose Client-Server Systems, *Proc. 7th Workshop on Hot Topics in Operating Systems*.
11. Aida, K., Takefusa, A., Nakada, H., Matsuoka, S., Sekiguchi, S. and Nagashima, U. (2000). Performance Evaluation Model for Scheduling in a Global Computing System, *The International Journal of High Performance Computing Applications*, **14**(3).
12. Foster, I. and Kesselman, C. (1997). Globus: A Metacomputing Infrastructure Toolkit, *The International Journal of Supercomputer Applications and High Performance Computing*, **11**(2): 115–128.
13. Natrajan, M.A., Humphrey and Grimshaw, A.S. (2001). Grids: Harnessing Geographically-Separated Resources in a Multi-organizational Context, *15th Annual International Symposium on High Performance Computing Systems and Applications*.
14. Casanova, H. (2001). Simgrid: A Toolkit for the Simulation of Application Scheduling, *Proc.1st IEEE/ACM International Symposium on Cluster Computing and the Grid*, Brisbane, Australia.
15. Van Steen, M., Homburg, P. and Tanenbaum, A. (1999). Globe: A Wide-area Distributed System, *IEEE Concurrency*, **7**(1): 70–78. http://www.cs.vu.nl/steen/globe/
16. Vadhiyar, S. and Dongarra, J. (2003). GrADSolve – A Grid-based RPC System for Remote Invocation of Parallel Software, *Journal of Parallel and Distributed Computing*, **63**(11): 1082–1104.
17. Casanova, H. and Dongarra, J. (1997). NetSolve: A Network-enabled Server for Solving Computational Science Problems, *The International Journal of Supercomputer Applications and High Performance Computing*, **11**(3): 212–223.
18. Edwards, W.K. (2000). *Core Jini,* **2nd edn**, New Jersey: Upper Saddle River, Prentice Hall.
19. Jini Architecture Specification. Available at URL: http://www.sun.com/jini/specs/jini1_1.pdf
20. Freeman, E., Hopfer, S. and Arnold, K. (1999). *Javaspaces™ Principles, Patterns, and Practice*, Massachusetts: Reading, Addison-Wesley.
21. Sobolewski (2002). *Federated P2P Services in CE Environments, Advances in Concurrent Engineering,* A.A. Balkema Publishers, pp. 13–22.
22. Zhao, S. and Sobolewski, M. (2001). Context Model Sharing in the FIPER Environment, *Proc. 8th Int. Conference on Concurrent Engineering: Research and Applications*, Anaheim, CA.

Dr. Sanjay Goel

Sanjay Goel is an Associate Professor in the school of business at the University at Albany, SUNY. He is also the director of research at the New York State Center for Information Forensics and Assurance at the University. Before joining the University, he worked at the General Electric Global Research Center. Dr. Goel received is PhD in mechanical engineering in 1999 from Rensselaer Polytechnic Institute. His current research interests include investigation of computer crimes including botnets and virus/worm propagation, security risk analysis and security policy creation. He also works in the development of autonomous computer security systems based on biological paradigms of immune systems, epidemiology, and genetics. His portfolio of research includes distributed service-based computing, network resilience, and active networks. He also uses machine-learning algorithms to develop self-learning adaptive optimization strategies for solving engineering optimization problems. In addition, he is working on developing algorithms for self-organization of nanosensors for remote sensing in harsh environments. Dr. Goel teaches several classes including computer networking and security, information security risk analysis, security policies, enterprise application development, database development and Java language programming. In 2006, he was awarded the SUNY Chancellor's Award for Excellence in Teaching, the University at Albany Excellence in Teaching Award, and the Graduate Student Organization Award for Faculty Mentoring.

Dr. M. Sobolewski

Dr M. Sobolewski joined, as a Professor, the Computer Science Department, Texas Tech University in September 2002. He is the Principal Investigator and Director of the SORCER laboratory focused on research in network, service, and distributed object-centric programming, and metaprogramming. While at GE Global Research Center he was the chief architect of the Federated Intelligent Product EnviRonment (FIPER) project, and developed other seventeen successful distributed systems for various GE business components. Prior to coming to U.S., during 18-year career with the Polish Academy of Sciences, Warsaw, Poland, he was the head of the Picture Recognition and Processing Department, the head of the Expert Systems Laboratory, and was doing research in the area of knowledge representation, knowledge-based systems, pattern recognition, image processing, neural networks, and graphical interfaces. He has served as visiting professor, lecturer and consultant in Sweden, Finland, Italy, Switzerland, Germany, Hungary, Czechoslovakia, Poland, Russia, and USA. He has over thirty years of experience in the development of large scale computing systems.

Dr. Talya

Dr Talya currently works as a Technical Leader in the Products Engineering group at GE Energy-Gasification. He joined GE Gasification in 2006. He currently leads a global team of 25 and is responsible for product development for Gasification/ IGCC related to Gasifier Design and Feed Injectors. Prior to this, Dr. Talya worked as a Project Leader at GE Global Research Center, Niskayuna, NY. He joined GE Global Research in 2000. He was involved with research and development in the area of structural design and multidisciplinary design optimization of Aircraft Engines and land-based gas turbines. His areas of expertise include Structural & Hydraulic Design of Mechanical Equipment, Design Automation Technologies and System Modeling & Analysis. He has worked with numerous GE Businesses including GE Aviation, GE Energy & GE Healthcare. Dr Talya obtained his Ph.D. in Mechanical Engineering from Arizona State University in Spring 2000. The primary focus of his Ph.D. was on Multidisciplinary Design Optimization of Cooled Gas Turbine blades for improved thermal and aerodynamic performance and Design Sensitivity Analysis of aerospace structures. He obtained his B.Tech. (BS) in Mechanical Engineering from Indian Institute of Technology, Chennai, India in 1996. Dr Talya is a Member of AIAA & ASME.