GLOBAL DETERMINISTIC AND STOCHASTIC OPTIMIZATION IN A SERVICE ORIENTED ARCHITECTURE

Chaitra Raghunath[a], Tyler Chang[a], Layne T. Watson[a,b,c],
Mohamed Jrad[c], and Rakesh K. Kapania[c]

[a]Department of Computer Science, [b]Department of Mathematics,
[c]Department of Aerospace & Ocean Engineering,
Virginia Polytechnic Institute & State University, Blacksburg, VA 24061
thchang@vt.edu

Raymond M. Kolonay

AFRL/RQVC, 2210 8th Street, Bldg. 146, Wright-Patterson Air Force Base, Dayton, OH 45433

ABSTRACT

Service ORiented Computing EnviRonment (SORCER) is a Java-based network-centric computing platform. SORCER provides a service oriented architecture, which enables the implementation of parallel algorithms in a dynamic distributed computing environment. SORCER is often used for multidisciplinary aircraft design analysis and optimization. Typically the distributed services are high-level disciplinary analyses such as structures, aerodynamics, controls, propulsion, etc. However, this approach often assigns intense optimization algorithms to run entirely on single overloaded nodes, rather than evenly distributing the workload. The goal of this work is to provide lower-level optimization algorithms, specifically VTDIRECT95 and QNSTOP, as integrated SORCER services and study the overhead of doing so. VTDIRECT95, a Fortran 95 implementation of D. R. Jones' algorithm DIRECT, is a highly parallelizable derivative-free deterministic global optimization algorithm. QNSTOP is a parallel quasi-Newton algorithm for stochastic optimization problems. The potential benefit of integrating VTDIRECT95 and QNSTOP into the SORCER framework is to provide dynamic load balancing among computational resources at the optimization level, resulting in a dynamically scalable process. Results are included for the multidisciplinary design optimization of an aircraft design application.

Keywords: service-oriented computing, deterministic global optimization, stochastic optimization, multi-disciplinary design

1 INTRODUCTION

This paper discusses the implementation and overhead of integrating two global optimization algorithms, VTDIRECT95 and QNSTOP, into a SORCER framework. SORCER is a large-scale, distributed computing environment for high fidelity multidisciplinary design optimization (MDO). The algorithms VTDIRECT95 and QNSTOP were chosen because of their relevance to aerospace engineering and their scalability on distributed computing applications. The process of integrating VTDIRECT95 and QNSTOP with SORCER is described in detail. The added overhead of the SORCER service is then assessed for an aircraft design application implementing VTDIRECT95 and QNSTOP on a SORCER grid.

Aerospace systems today exhibit strong interdisciplinary interactions and require a multidisciplinary, collaborative approach (Raymer 2006). Multidisciplinary design optimization aims to achieve an optimal design over several disciplines. The first step in the design process, conceptual design, often requires optimization with a large number of design variables belonging to multiple disciplines. Traditional conceptual design is carried out over a large set of configurations with low fidelity models, and suffers from poor accuracy. However, new physics based modeling tools used with high end computing resources can provide accurate multiphysics analysis and in the early stages of design. The drawback of these high fidelity models is that they are often prohibitively complex.

High performance computing (HPC) systems are critical for complex large scale design studies (Kodiyalam et al. 2004). While HPC systems deliver high computational power (capability computing) or high throughput (capacity computing), they are static resources with little scalability or flexibility. Service-oriented architecture (SOA) addresses the challenges faced by HPC systems in terms of scalability, availability, flexibility, and reliability. SOA not only incorporates the features of HPC systems, but also promises a world of orchestrated services by creating dynamic processes and agile applications that span platforms and organizations(Georgakopoulous and Papazoglou 2008).

SORCER, a Java based network centric computing framework (maintained by SORCERsoft.com, a subsidiary of SMT S. A. group), is a federated service-to-service (S2S) metacomputing environment that treats service providers as network peers with well-defined semantics of a federated service object-oriented architecture (Raghunath 2015). SORCER provides a platform for high fidelity multidisciplinary design optimization, combining models from various disciplines into one integrated model. SORCER accommodates dynamic distribution of service providers and on-demand provisioning of resources, resulting in significant speedups and effective utilization of computational resources.

VTDIRECT95 and QNSTOP were chosen for implementation on a SORCER grid because of their relevance to aircraft design problems. VTDIRECT95, a massively parallel Fortran 95 implementation of D. R. Jones' algorithm DIRECT, is widely used in MDO (Gao et al. 2013, Ghommem et al. 2012, Mehmood et al. 2011). QNSTOP is a class of parallel quasi-Newton methods for stochastic optimization and deterministic global optimization. QNSTOP for stochastic optimization problems synthesizes ideas from numerical optimization and response surface methodology, and demonstrates potential for stochastic robust design optimization and stochastic MDO problems.

The paper is organized as follows. Section 2 outlines the SORCER framework and the two optimization algorithms, VTDIRECT95 and QNSTOP. Section 3 presents details about conversion of VTDIRECT95 and QNSTOP to SORCER services. The study of an aircraft design application using VTDIRECT95 and QNSTOP as SORCER services is presented in Section 4. Section 5 briefly discusses the results of this work.

2 BACKGROUND

2.1 SORCER Overview

Service-oriented computing is a computing paradigm that utilizes self-describing, platform-agnostic services as the fundamental constructs to support rapid, cost-effective composition of distributed applications (Papazoglou et al. 2007). Services are self-adapting, dynamic processes that effectively communicate with one another to perform user-requested tasks in a distributed computing environment. The service-oriented computing paradigm, derived from the SOA model, allows interoperability, reusability, and loose coupling of its components in a dynamic environment, where computer resources are assigned to services as and when necessary. As indicated in Figure 1, the interaction between software agents is facilitated by message exchanges between service providers and service requestors. The service provider determines a description

for a service and publishes it to a service discovery agency. This, in turn, is made discoverable to a service requestor. To invoke a service, the service requestor retrieves the service description from a registry and binds with the service provider based on the service description. In short, SOA addresses the challenges of distributed computing by enabling service discovery, integration, and use (Georgakopoulous and Papazoglou 2008).

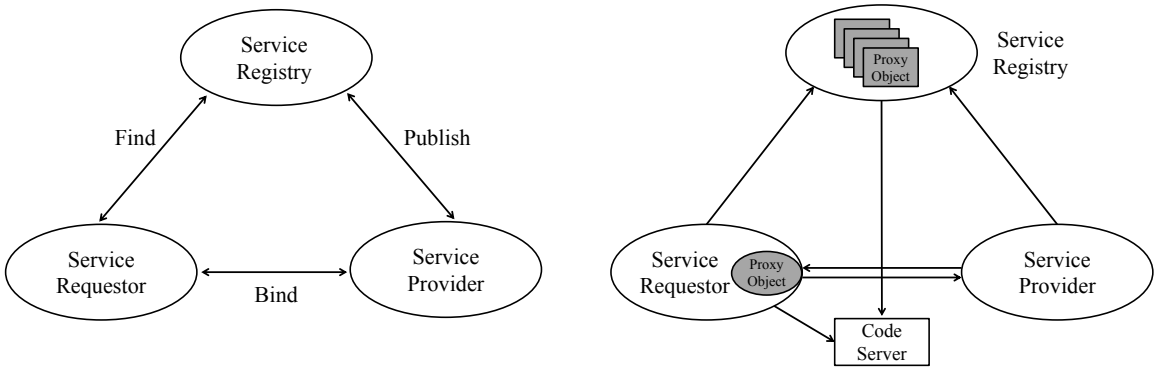

Figure 1: Service-oriented architecture (SOA) (left) vs. service object-oriented architecture (SOOA) (right).

SORCER is based on the concepts of SOA and also incorporates features of the service object-oriented architecture (SOOA), where service providers are objects accepting remote invocations (Raghunath 2015). As shown in Figure 1, the service requestor binds to the service provider by creating a proxy for remote communication. SOOA permits great flexibility in terms of communication between agents. These proxies, known as smart proxies, grant access to local and remote resources, regardless of who initially created the proxy. In SORCER, providers broadcast their availability, registries intercept broadcasted announcements and cache proxy objects to their service providers (Raghunath 2015). The SORCER operating system (SOS) looks up proxies by sending queries to registries and making selections from the available services. In short, providers use discovery/join protocols to publish services in the network, and SOS uses discovery/join protocols to obtain services in the network. From an object-oriented programming point of view, service providers are represented as independent network objects, locating each other via service registries and communicating through protocols such as remote method invocation (RMI), simple object access protocol (SOAP), common object request broker architecture (COBRA), etc.

Further, SORCER introduces three layers of converged programming abstractions: exertion-oriented programming (EOP), var-oriented programming (VOP), and var-oriented modeling (VOM) (Raghunath 2015). The EOP abstraction manages object-oriented distributed system complexity introduced by the complex network of metacomputers. VOP is a paradigm based on dataflow principles where changing the value of a var automatically forces recalculation of the interdependent values of vars. VOM, a modeling paradigm using vars, defines heterogeneous multidisciplinary var-oriented models in large scale multidisciplinary models. Thus, the SORCER framework incorporates the power of object-oriented programming and exertion-oriented programming to create an infrastructure that is modular, extensible, and reusable.

Based on successful implementation of large scale engineering applications with SORCER (Raghunath 2015), several desirable features related to multidisciplinary aircraft analysis and design optimization are as follows:

- Large scale, distributed, decentralized: SORCER dynamically federates processes and smartly distributes the load across all machines in the network.
- Leveraging the power of HPC: SORCER provides the features and computing power of HPC and SOA to form a dynamic distributed engineering collaboration platform.
- Reusability: The incorporation of object-orient modularity enables a high level of reuse when moving from one study to the next.
- Cost effective: SORCER accommodates physics based modeling via HPC for faster evaluation of higher fidelity configurations at the preliminary level of design when compared to traditional practices.
- Better utilization of computational resources: SORCER enables collaborative design studies across organizational boundaries and maximum utilization of all compute resources on the network, ranging from personal computers to high performance computing machines.
- Distributed resource management: SORCER employs Jini Connection technology (now called Apache River) with its JavaSpaces service to implement computational resource management across the network. The JavaSpaces technology facilitates the implementation of a self-load limiting grid computing system that can dynamically grow and shrink during the course of an optimization study (Freeman, Hupfer, and Arnold 1999). The loosely coupled space-based service federation allows asynchronous communication between computers in the network in a reliable manner (Raghunath 2015).

2.2 VTDIRECT95

VTDIRECT95 is a Fortran 95 software package using massively parallel dynamic data structures to implement the algorithm DIRECT by Jones, Perttunen, and Stuckman (1993). The algorithm DIRECT (DIviding RECTangles) is a deterministic global optimization algorithm that performs Lipschitzian optimization without the Lipschitz constant, and can be classified as a derivative free direct search algorithm.

Let E^n denote real n-dimensional Euclidean space, $D = \left\{x \in E^n \mid \ell \leq x \leq u\right\}$ be a box in E^n, and $f : D \to E$ a Lipschitz continuous function. The problem is to find a global minimum point $\bar{x}$ of f over D, $f(\bar{x}) = \min_{x \in D} f(x)$. The original (serial) algorithm by Jones, Perttunen, and Stuckman (1993)is described in six steps as below:

Step 1 (initialization): Normalize the feasible set D to be the unit hypercube. Sample the center point c_i of this hypercube and evaluate $f(c_i)$. Initialize $f_{min} := f(c_i)$, evaluation counter $m := 1$, and iteration counter $t := 0$.

Step 2 (selection): Identify the set S of "potentially optimal" boxes (subregions) of D. A box is potentially optimal if, for some Lipschitz constant, the function value within the box is potentially smaller than that in any other box (a formal definition with parameter ϵ is given by Jones, Perttunen, and Stuckman (1993)).

Step 3 (sampling): For any box $j \in S$, identify the set I of dimensions with the maximum side length. Let δ equal one-third of this maximum side length. Sample the function at the points $c \pm \delta e_i$ for all $i \in I$, where c is the center of the box and e_i is the ith unit vector.

Step 4 (division): Divide the box j containing c into thirds along the dimensions in I, starting with the dimension with the lowest value of $w_i = \min\{f(c + \delta e_i), f(c - \delta e_i)\}$, and continuing to the dimension with the highest w_i. Update f_{min} and m.

Step 5 (iteration): Set $S := S \setminus \{j\}$. If $S \neq \emptyset$, go to Step 3.

Step 6 (termination): Set $t := t + 1$. If iteration limit or evaluation limit has been reached, stop. Otherwise, go to Step 2.

VTDIRECT95 has numerous modifications from DIRECT in order to improve performance and load balancing on large scale parallel systems. The massively parallel implementation VTDIRECT95 distributes data among processors to share the memory burden imposed by storing all current boxes. The parallel scheme for SELECTION concentrates on distributing data among multiple masters to share the memory burden. Functional parallelism for SAMPLING is achieved by fully distributed control allocating function evaluation tasks to workers. A detailed discussion of the implementation of the serial and parallel subroutines in VTDIRECT95 is presented in He, Watson, and Sosonkina (2009).

2.3 QNSTOP

QNSTOP is a class of quasi-Newton methods for stochastic optimization with variations for deterministic global optimization (Amos et al. (2014b)). In iteration k, QNSTOP methods compute the gradient vector $\hat{g}_k$ and Hessian matrix $\hat{H}_k$ of a quadratic model

$$\hat{m}_k(X - X_k) = \hat{f}_k + \hat{g}_k^T(X - X_k) + \frac{1}{2}(X - X_k)^T \hat{H}_k(X - X_k)$$

of the objective function f centered at X_k, where $\hat{f}_k$ is generally not $f(X_k)$. QNSTOP methods progress by

$$X_{k+1} = \left[X_k - [\hat{H}_k + \mu_k W_k]^{-1} \hat{g}_k\right]_\Theta,$$

where μ_k is the Lagrange multiplier of a trust region subproblem, W_k is a scaling matrix, and $[\cdot]_\Theta$ denotes projection on the feasible set Θ. The exact steps taken at each iteration are outlined below:

Step 0 (initialization): Given a function evaluation budget $\widetilde{B}$ per start point and operating mode (deterministic or stochastic), set values for $\tau_0 > 0$, $\mu_0 > 0$, $\gamma \geq 1$, $\eta \geq 0$, $\zeta \geq 0$, N, X_0, $k := 0$, $W_0 := \hat{H}_0 := I_p$. It is recommended to run QNSTOP multiple times from different starting points.

Step 1 (regression experiment): Compute the ellipsoidal design regions given by

$$E_k(\tau_k) = \left\{X \in E^p : (X - X_k)^T W_k (X - X_k) \leq \tau_k^2\right\}$$

where τ_k is decayed at some rate depending on the mode. Next, uniformly sample $\{X_{k1}, \ldots, X_{kN}\} \subset E_k(\tau_k) \cap \Theta$, where Θ denotes the feasible set, and observe the response vector Y_k, where y_{ki} is modeled by $y_{ki} = \hat{f}_k + X_{ki}^T \hat{g}_k + \epsilon_{ki}$, with ϵ_{ki} accounting for the lack of fit. Finally, compute the least squares estimate for the gradient $\hat{g}_k$ using

$$\left(D_k^T D_k\right) \hat{g}_k = D_k^T Y_k.$$

where D_k denotes the absolute deviations of X_{ki}.

Step 2 (secant update): If $k > 0$, compute the model Hessian matrix $\hat{H}_k$ using BFGS (deterministic) or SR1 variant (stochastic) update.

Step 3 (update iterate): Calculate the next iterate X_{k+1}. In the deterministic case, X_{k+1} is the solution to the optimization problem

$$\min_{X \in E_k(\rho_k)} \hat{g}_k^T(X - X_k) + \frac{1}{2}(X - X_k)^T \hat{H}_k(X - X_k).$$

In the stochastic case, X_{k+1} is obtained by directly updating the Lagrange multiplier μ_k as described in Castle (2012), using the update:

$$X_{k+1} = X_k - \left[\hat{H}_k + \mu_k W_k\right]^{-1} \hat{g}_k$$

In both cases the computed point X_{k+1} is projected onto the feasible set Θ.

Step 4 (update subsequent design ellipsoid): Compute an updated scaling matrix $W_{k+1} \in W_\gamma$ as described in Castle (2012)and Amos et al. (Amos et al. (2014b)).

Step 5: If $(k+2)(N+1)+1 < \widetilde{B}$ then increment k by 1 go to **Step 1**. Otherwise, the algorithm terminates. (f is also observed at each ellipsoid center X_k.)

The algorithm QNSTOP has three significant sources of parallelism: the individual function evaluations, the loop over the samples in an experimental design, and the loop over the start points. A master-slave paradigm is a reasonable approach if the individual function evaluations are large scale parallel simulations. On large shared memory systems, ample parallelism is exhibited at the two outer nested loops — the loop over the start points and the loop over the samples. A detailed discussion of the serial and parallel implementations of QNSTOP can be found in Amos et al. (2014b). An analysis of a serial Fortran 95 implementation of QNSTOP is presented in Amos et al. (2014a).

2.4 Discussion

In the context of ever increasing parallelism, higher dimensions, and multidisciplinary design optimization, algorithms like VTDIRECT95 (for deterministic global optimization) and QNSTOP (for stochastic optimization) are excellent candidates for SORCER services. Objective function cost is one of the key parameters that affects the parallel performance under different parallel schemes. High parallel efficiency involves balancing communication overhead with the distribution of evaluation tasks for good load balancing (He et al. 2009a, He et al. 2009b). While SORCER has no control of the definition and granularity of the tasks, it *can* provide robust distributed parallelization and load balancing across computational resources, thus significantly speeding up the evaluation of objective functions in a dynamically scalable metacomputing environment.

3 IMPLEMENTATION AS SORCER SERVICES

3.1 JNI Wrappers

SORCER leverages the power of distributed computing through the use of Java interoperability, Jini, and web services (Raghunath 2015). The adoption of Java as a language for numerical computing presents difficulties. Some obstacles include: overrestrictive floating point semantics, inefficient support for complex numbers and alternative arithmetic systems, and lack of direct support for true multidimensional arrays (Boisvert et al. 2001). Moreover, the task of manually converting existing code in Fortran to Java-based services is both daunting and expensive (Liang 1999).

The Fortran 95 implementations of optimization algorithms considered in this paper, VTDIRECT95 and QNSTOP, incorporate advanced Fortran features that flexibly organize the data on a single machine, effectively reduce the local data storage, and efficiently share the data across multiple processors (He, Watson, and Sosonkina 2009). While Fortran is effective for numerical computing, Java provides flexibility and scalability for dynamic grid-based network architectures. In order to cope with the heterogeneity imposed by various programming languages, Java wrappers for the existing legacy code have been implemented using the JNI (Java Native Interface) libraries.

In developing the wrappers for the existing Fortran 95 implementations of VTDIRECT95 and QNSTOP, a feature of the JNI called the *invocation interface* was used. The invocation interface allows a regular non-Java program running on the native operating system to invoke a JVM to gain access to Java classes and features (Liang 1999). The invocation interface allows developers to embed a JVM implementation into native applications. Native applications can link with a native library that implements the JVM, and then use the invocation interface to execute components written in the Java programming language (Lindsey, Tolliver,

and Lindblad 2010). Further, a C or C++ layer (often referred to as "glue code") is required to gain access to codes written in Fortran.

3.2 Design Analysis with SORCER services

The concept of a service provider, or simply 'provider', is the crux of an engineering analysis or design study using SORCER. A provider is Java code that makes a number of Java methods (services) available to users over a network. Each provider is implemented in accordance with the principles of exertion-oriented programming (EOP), where an *exertion* is an object that represents a process by specifying the relationship between services and the information passed between them.

A provider is published on the network using SORCER. The provider's service may then be accessed via a small Java code called a *service requestor*. An individual request running on a single provider is called a *Task*. The input/output data associated with a task exectution is called a *Context*.

Providers are of two kinds — *analysis providers* and *model providers*. Providers that leverage existing domain-specific codes are referred to as analysis providers (Burton, Alyanak, and Kolonay 2012). An 'analysis provider' is an entity that neatly wraps the underlying domain-specific code with Java code so the domain-specific code can be accessed as a service by a remote user. The domain-specific code is generally platform independent and performs the bulk of the engineering-specific computations for a given service. Model providers are explained in the context of optimization problems over the remainder of this section.

In SORCER terminology, a *model* is a collection of service-oriented variables called *vars*. A var is defined by a triplet $\langle$*value, evaluator, filter* $\rangle$, where

- a *value* is an expression yielding a valid quantity;
- an *evaluator* defines the process of how data is produced via remote services, or produced locally;
- a *filter* reduces the data generated by the evaluator to the value of the var.

In this way, SORCER leverages the power of var-oriented programming (VOP) to handle large sets of interconnected variables and does so in accordance with the var-based modeling paradigm var-oriented modelling (VOM).

In the context of optimization, these vars are the design variables and the implementation of the objective and constraint functions. Var instances are used to model both independent and dependent variables in SORCER. While independent vars are used as a container to store a value and perform no calculations, dependent vars implement mathematical functions. When published on the network, these dependent and independent vars that define a specific optimization problem are referred to as a *model provider*. The model provider is characterized by a single state and behaves like shared memory to users over the network.

For each objective function evaluation, a query object containing the name of the model provider, the design variable var names and values, and the var names of the objective function that the user wishes to calculate is constructed. The corresponding published model provider receives the query object and invokes the *setValue* method on its design variables and subsequently the *getValue* method on the user-specified objective function. The query object is then returned to the user with the updated values. To obtain the most recently updated value of the dependent var (the user-specified objective function), the model invokes the *evaluator* function. The evaluator checks the values of its arguments every time it is called, and will only recalculate its dependent vars if it detects a change.

3.3 Subroutines as a SORCER service

3.3.1 Serial Subroutines

For the serial subroutines VTdirect and QNSTOPS, platform independent executables are implemented using JNI (as described in Section 3.1) and tightly coupled with the provider's service.

3.3.2 pVTdirect

Unfortunately, results for pVTdirect (the massively parallel implementation of DIRECT in the package VTDIRECT95) under SORCER are not presented here because pVTdirect is fundamentally incompatible with efficient usage of the SORCER/JavaSpace/table model query paradigm (described in Sections 3.3.3 and 4.1). This paradigm assumes a master-slave parallel computing paradigm, and is not valid for a fully distributed algorithm such as pVTdirect.

3.3.3 QNSTOPP

The parallel (OpenMP) implementation (subroutine QNSTOPP) of QNSTOP incorporates three sources of parallelism: (1) the loop over the start points, (2) the loop over the experimental design samples, or (3) both. For compatibility with SORCER, QNSTOPP is modified at the level of the loop over the experimental design samples such that the function evaluations are chunked in function evaluation calls to SORCER. In this case, QNSTOPP interacts with the published model provider via a table model query. Rather than passing a single design point to the model provider, the JNI wrapper constructs a table containing the name of the design vars and their values for a set of sample points. As with the case of a single model query, a query object containing the name of the model provider, the design variable var names and values, and the var names of the objective function is constructed. The model provider, on receiving the query object, creates new child instances for each row in the table for parallel execution of the table row evaluations. The model provider creates a thread for each child instance and begins to *setValue* and *getValue* on the vars. On completion, the var values for each run are returned to the JNI wrapper in a table object and the child models are discarded.

4 EXPERIMENTS AND RESULTS

The framework *EBF3PanelOpt* facilitates the structural optimization of curvilinearly stiffened panels by considering a number of constraints that have to be satisfied (buckling, von Mises stress, and crippling constraints). The framework, written in Python, interacts with the commercial software MSC Patran (for geometry and mesh creation) and MSC Nastran (for finite element analysis). Given the input parameters and design variables, the script then creates the appropriate session file and submits it to MSC Patran to create the geometry and mesh of the stiffened panel, with which MSC Nastran then carries out a finite element analysis producing the inputs to an optimizer. More details about the framework EBF3PanelOpt can be found in Mulani, Slemp, and Kapania (2013). By decomposing an aircraft wing into multiple local panels bordered with curvilinear spars and ribs, aircraft designers can utilize EFB3PanelOpt to minimize the structural weight of these panels and subsequently the overall wing weight.

This section presents the implementation and results for optimization of curvilinear blade-stiffened panels using VTDIRECT95 and QNSTOP. The section is further divided into two subsections — the Implementation of EBF3PanelOpt as a SORCER service, and the performance results for the optimization of a stiffened panel.

All experiments presented here are conducted on Intel (i7-3770) machines running at 3.4 GHz, each with 16GB of memory and a single quad-core processor, in which each core supports hyperthreading. The experiments are conducted using GNU Fortran 4.9.1, GNU C 4.9.1, Python 2.6.6, Open MPI 1.8.1, and Java 1.8.0_25 on x86_64 running CentOS 6.6. The framework EBF3PanelOpt is configured to use Nastran 2014

and Patran 2014. For all QNSTOPP runs with SORCER presented in this paper, two identical machines are used. The program carrying out optimization as a service and the model provider are started on one machine, and the EBF3PanelOpt provider on the other.

4.1 EBF3PanelOpt as a Service

In order to achieve truly distributed objective function evaluations, the optimization framework EBF3Panel-Opt is implemented as an analysis provider. Analysis providers can be dynamically distributed over a variety of computational resources with the help of SORCER's JavaSpaces technology. The JavaSpaces technology provides a type of shared memory where exertion evaluators can drop tasks to be processed by service providers, which use Jini discovery mechanisms to find them on the network. JavaSpaces not only enables computers on the network to communicate reliably, but also provides load balancing capability to cope with dynamic resources. Hence, computing resources can be added during the course of an optimization study, thereby enhancing productivity.

For the parallel implementation of QNSTOP that constructs a table of runs (a modification of the OpenMP parallel code QNSTOPP), the EBF3PanelOpt provider is configured to have a fixed number of worker threads. The number of worker threads determines the number of tasks a provider can process in parallel. The providers are started on multiple machines to distribute the work. During optimization, the model provider first creates child instances for every row in the table and drops these tasks into the space. Then, based on the number of worker threads, each EBF3PanelOpt provider picks up unprocessed tasks from the space and executes them in parallel.

An undocumented alternative to JavaSpaces is Catalog, referred to here as SORCER/Catalog, that has advantages in certain contexts, such as multicore or single large parallel distributed memory machines. Here, service providers publish proxies to the catalog. The requestor passes a service request to the catalog, which "matches" the service request with one of the proxies, which is then passed to the requestor, who uses the proxy to make the remote call directly to the provider (as in Figure 1).

A detailed description of the implementation is presented in Raghunath (2015)with an experiment that recreates the designs reported in Mulani, Slemp, and Kapania (2013).

4.2 Experiment

In this section, a simply supported flat rectangular panel is optimized for minimum mass using VTDIRECT95 and QNSTOP. The performance results for the optimization of curvilinear blade-stiffened panels containing two stiffeners (Case 1) and four stiffeners (Case 2), using the subroutines VTdirect, pVTdirect, QNSTOPS, and QNSTOPP with and without SORCER are presented below. The design variables for both problems and the exact settings used with the subroutines VTdirect, pVTdirect, QNSTOPS, and QNSTOPP are presented in Raghunath (2015).

For optimization with (the serial subroutine) VTdirect and (the parallel subroutine) pVTdirect, the stopping condition is a limit of 1000 on the number of objective function evaluations. For pVTdirect, the number of processes is set to 4. Since pVTdirect is incompatible with SORCER/JavaSpace/table model query (as described in Section 3.3.2), results for these cases are omitted.

For optimization with (the serial subroutine) QNSTOPS, deterministic mode is used with 5 start points and a budget of of 200 evaluations each (1000 total). Additionally, for JNI runs without SORCER, (the parallel subroutine) QNSTOPP is parallelized over just the sampling point objective function evaluations with 4 OpenMP threads. Recall from section 3.3.2, QNSTOPP for SORCER does not use OpenMP to achieve parallelism, but instead interacts with the model provider via a table model query. For all QNSTOPP runs, the table is configured with four rows, resulting in 4 concurrent objective function evaluations.

Table 1. Execution time in seconds for pylon wing panel optimization with 2 stiffeners

	VTdir	pVTdir	QNSTOPS	QNSTOPP	E_p
SORCER and script robustness	13,009	N/A	11,388	3,545	0.80
SORCER w/o script robustness	8,957	N/A	7,994	2,542	0.79
SORCER/Catalog w/o script robust.	8,487	N/A	7,597	2,458	0.77
W/o SORCER, w/o script robust.	8,460	2,924	7,560	2,309	0.82

Table 2. Execution time in seconds for pylon wing panel optimization with 4 stiffeners

	VTdir	pVTdir	QNSTOPS	QNSTOPP	E_p
SORCER w/ script robustness	14,450	N/A	10,370	3,676	0.71
SORCER w/o script robustness	10,384	N/A	7,451	2,697	0.69
SORCER/Catalog w/o script robust.	9,815	N/A	7,088	2,615	0.68
W/o SORCER, w/o script robust.	9,786	3,789	7,052	2,408	0.73

To calculate the parallel efficiency of QNSTOPP with and without SORCER, the parallel efficiency measure E_p was used. For the runs that do not involve SORCER

$$E_p = \frac{\big((\text{QNSTOPS time})/(\text{QNSTOPP time})\big)}{(\text{total number of OMP threads})}.$$

For the runs with SORCER

$$E_p = \frac{\big((\text{QNSTOPS time})/(\text{modified QNSTOPP time})\big)}{(\text{number of function evaluation threads})}.$$

In Tables 1, 2, and 3 "script robustness" refers to a Java utility `GenericUtil` separate from SORCER, recommended for use of scripts in production distributed computing, that increases the robustness of scripts and communication links across different operating systems.

4.2.1 Performance Results

The execution times for pVTdirect, VTdirect, QNSTOPS, and QNSTOPP, with and without SORCER, are listed in Tables 1 (Case 1) and 2 (Case 2). The last column in the table represents the parallel efficiency with QNSTOPP.

In Case 2, note that the numbers of function evaluations for VTdirect and QNSTOP* (1165 and 825 respectively) are different than those for Case 1 (1131 and 950 respectively). Even though the problem size doubled (from 13 to 25), the parallel efficiencies decreased because the number of function evaluations by QNSTOPP was less for Case 2 than for Case 1.

For 100 runs of VTdirect, the average computational expense of each objective function evaluation is listed in Table 3, where n is the problem dimension. The SORCER with script robustness overhead per function evaluation ($\approx 4\,s$) is about the same for both problem sizes, and without script robustness the SORCER overhead is negligible. Robustness and portability do not come cheap.

Table 3. Objective function evaluation times in seconds for pylon wing panel (2 & 4 stiffeners)

	$n = 13$	$n = 25$
With SORCER and script robustness	11.13	12.90
With SORCER, without script robustness	7.36	9.14
Without SORCER and script robustness	7.32	9.10

5 DISCUSSION

VTDIRECT95 and QNSTOP were implemented as services on a SORCER grid, facilitating the optimization of curvilinearly stiffened panels in a truly distributed manner. Source code for the service wrappers is in Raghunath (2015). From Table 3, the SORCER with script robustness overhead is about four seconds (essentially all due to script robustness). Tables 1 and 2 show that the other SORCER overhead is not significant for expensive function evaluations. The conclusion is that the right SORCER paradigms (JavaSpaces, though the most general approach, is but one of several) must be used in the right contexts, and no single SORCER paradigm is a general purpose solution to parallel and distributed computing for MDO. On the continuum of distributed computing technology (MPI, Globus, Legion, Condor, SORCER), SORCER is at the heavyweight end.

The use of JNI to wrap the corresponding native code provided a clean and elegant interface between the optimization algorithm and the Java block that evaluates the objective for a design point. However, the JNI development overhead is relatively high. It should also be mentioned that the installation of SORCER is far from routine—SORCER is currently a research code with limited documentation and requires considerable knowledge of network computing to install and utilize.

ACKNOWLEDGEMENTS

This material is based on research sponsored by Air Force Research Laboratory under agreement number FA8650-09-2-3938. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes not withstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of Air Force Research Laboratory or the U.S. Government. The EBF3PanelOpt code was developed under a research contract from NASA Fundamental Aeronautics Program to Virginia Polytechnic Institute and State University with Karen M. B. Taminger as the Program Manager.

REFERENCES

Amos, B. D., D. R. Easterling, L. T. Watson, B.S. Castle, M. W. Trosset, and W. I. Thacker. 2014a. "Fortran 95 Implementation of QNSTOP for Global and Stochastic Optimization". In *22nd High Performance Computing Symposium (HPC 2014)*. Tampa, Florida, Society for Computer Simulation International.

Amos, B. D., D. R. Easterling, L. T. Watson, W. I. Thacker, B. S. Castle, and M. W. Trosset. 2014b. "Algorithm XXX: QNSTOP: Quasi-Newton Algorithm for Stochastic Optimization". Technical Report 2014-07, Virginia Polytechnic Institute and State University, Blacksburg, VA.

Boisvert, R. F., J. Moreira, M. Philippsen, and R. Pozo. 2001. "Java and Numerical Computing", *IEEE Computing in Science and Engineering* vol 3(2), pp. 18–24.

Burton, S. A., E. J. Alyanak, and R. M. Kolonay. 2012. "Efficient Supersonic Air Vehicle Analysis and Optimization Implementation using SORCER". In *AIAA 2012- 5520, 12th AIAA Aviation Technology, Integration, and Operations (ATIO) Conference and 14th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*. Indianapolis, Indiana, American Institute of Aeronautics and Astronautics and International Society for Structural and Multidisciplinary Optimization.

Castle, B. S.. 2012. *Quasi-Newton Methods for Stochastic Optimization and Proximity-based Methods for Disparate Information Fusion*. Ph.D. thesis, Indiana University, Bloomington, IN.

Freeman, E., S. Hupfer, and K. Arnold. 1999. *JavaSpaces Principles, Patterns, and Practice*. Boston, MA, Addison Wesley Longman, Inc.

Gao, D. Y., L. T. Watson, D. R. Easterling, W. I. Thacker, and S. C. Billups. 2013. "Solving the Canonical Dual of Box- and Integer-constrained Nonconvex Quadratic Programs via a Deterministic Direct Search Algorithm", *Optimization Methods and Software* vol. 28(2), pp. 313–326.

Georgakopoulous, D. and M. P. Papazoglou. 2008. *Service-Oriented Computing*. Cambridge, The MIT Press.

Ghommem, M., M. R. Hajj, B. K. Stanford, L. T. Watson, and P. S. Beran. 2012. "Global and Local Optimization of Flapping Kinematics". In *53rd AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference*. Honolulu, Hawaii, American Institute of Aeronautics and Astronautics.

He, J., A. Verstak, M. Sosonkina, and L. T. Watson. 2009a. "Performance Modeling and Analysis of a Massively Parallel DIRECT: Part 2", *International Journal of High Performance Computing Applications* vol. 23(1), pp. 29–41.

He, J., A. Verstak, L. T. Watson, and M. Sosonkina. 2009b. "Performance Modeling and Analysis of a Massively Parallel DIRECT: Part 1", *International Journal of High Performance Computing Applications* vol. 23(1), pp. 14–28.

He, J., L. T. Watson, and M. Sosonkina. 2009. "Algorithm 897: VTDIRECT95: Serial and Parallel Codes for the Global Optimization Algorithm DIRECT", *ACM Transactions on Mathematical Software (TOMS)* vol. 36(3), Article No. 17.

Jones, D. R., C. D. Perttunen, and B. E. Stuckman. 1993. "Lipschitzian Optimization without the Lipschitz Constant", *Journal of Optimization Theory and Application* vol. 79(1), pp. 157–181.

Kodiyalam, S., R. J. Yang, L. Gu, and C. H.. 2004. "Multidisciplinary Design Optimization of a Vehicle System in a Scalable, High Performance Computing Environment", *Structural and Multidisciplinary Optimization* vol. 26(3/4), pp. 256–263.

Liang, S.. 1999. *The Java Native Interface: Programmer's Guide and Specification*. MA, Addison Wesley Longman Inc.

Lindsey, C. S., J. S. Tolliver, and T. Lindblad. 2010. *JavaTech, an Introduction to Scientific and Technical Computing with Java*. Cambridge, Cambridge University Press.

Mehmood, A., I. Akhtar, M. Ghommem, M. R. Hajj, and L. T. Watson. 2011. "Optimization of Drag Reduction on a Cylinder Undergoing Rotary Oscillations". In *52nd AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference*. Denver, Colorado, American Institute of Aeronautics and Astronautics.

Mulani, S. B., W. C. H. Slemp, and R. K. Kapania. 2013. "EBF3PanelOpt: an Optimization Framework for Curvilinear Blade-stiffened Panels", *Thin Walled Structures* vol. 63, pp. 13–26.

Papazoglou, M. P., P. Traverso, S. Dustdar, and F. Leymann. 2007. "Service-Oriented Computing: State of the Art and Research Challenges", *Computer* vol. 40(11), pp. 38–45.

Raghunath, C.. 2015. *Service Oriented Computing Environment (SORCER) for Deterministic Global and Stochastic Optimization*. M.S. thesis, Department of Computer Science, Virginia Polytechnic Institute & State University, Blacksburg, VA.

Raymer, D. P.. 2006. *Aircraft Design: A Conceptual Approach*. New York, American Institute of Aeronautics and Astronautics Education Series.