Proceedings of the 2006 Winter Simulation Conference
L. F. Perrone, F. P. Wieland, J. Liu, B. G. Lawson, D. M. Nicol, and R. M. Fujimoto, eds.

ON THE PERFORMANCE OF INTER-ORGANIZATIONAL DESIGN OPTIMIZATION SYSTEMS

Paolo Vercesi

ESTECO
99, Padriciano
Trieste, TS 34012, ITALY

Alberto Bartoli

DEEI
University of Trieste
Trieste, TS 34127, ITALY

ABSTRACT

Simulation-based design optimization is a key technology in many industrial sectors. Recent developments in software technology have opened a novel range of possibilities in this area. It has now become possible to involve multiple organizations in the simulation of a candidate design, by composing their respective simulation modules on the Internet. Thus, it is possible to deploy an *inter-organizational design optimization* system, which may be particularly appealing because modern engineering products are assembled out of smaller blocks developed by different organizations. In this paper we explore some of the fundamental performance-related issues involved in such a novel scenario, by analyzing a variety of options: centralized control vs. distributed control; generation of new candidate designs one at a time or in batches; communication and computation performed serially or with time overlap. Our analysis provides useful insights into the numerous trade-offs involved in the implementation of inter-organizational design optimization.

1 INTRODUCTION

Design optimization is one of the key problems to be faced in nearly all industrial sectors. Optimizing, or simply improving, the design of a component or a system is a complex and time-consuming job, but it is also a necessity to stay competitive. Satisfying this requirement while achieving faster product development times, faster product innovation and turnaround, lower costs is becoming more and more difficult. In many cases, the use of specialized simulation software has become the only practical way for addressing such conflicting goals. In particular, *simulation-based optimization* systems (Swisher et al. 2000, Banks et al. 2005, April et al. 2003) and *design-optimization* environments allow exploring efficiently the design search space: they generate a set of candidate designs, evaluate each design by means of simulation and then generate a new set of better designs, based on some search optimization strategy. The simulation techniques used for evaluating each "virtual prototype" include disparate applications ranging from parametric CAD, to Finite Element Solvers, Discrete Event Simulators software, legacy solvers. The entire process proceeds automatically and iterates until one or more satisfactory designs are found. At this point, the results are given to human specialists, e.g., a team of engineers, for further analysis and evaluation.

This approach is now widely used and has become a key component of many CAD/CAE tools. Concerning the specific system with which we are more familiar (<http://www.esteco.com>), significant successful applications have been performed in a number of different sectors: aerospace and defense (Gaiddon, Knight, and Poloni 2004), appliances, architectural, automotive (Fu, Kachnowski, and Lee 2004), biomedical (Taga, Funakubo, and Fukui 2005), experimental data, food and beverage, manufacturing, marine and off–shores (Maisonneuve et al. 2003; Boulougouris, Papanikolaou, and Zaraphonitis 2004), turbomachinery and casting (Hepp, Lohne, and Sannes 2003). Other applications can be found in multi-disciplinary design optimization (Giassi, Bennis, and Maisonneuve 2004; Botros et al. 2004).

Recent developments in software technology have opened a new range of possibilities for simulation-based design optimization systems (hereinafter, design optimization systems for short). Technologies capable of greatly simplifying the integration among remote programs hosted by different organizations are now widely available. Web service technology (Gottschalk et al. 2002, Stal 2002, Alonso et al. 2004), for example, allows an organization to expose (some of) the functionality of its internal systems on the Internet and to make it discoverable and accessible through the World Wide Web in a controlled manner. It has thus become possible to build inter-organizational services by combining multiple services exported by single organizations. As a result, one can build complex services resulting from the Internet-based integration of simpler building blocks, possi-

1-4244-0501-7/06/$20.00 ©2006 IEEE

bly consisting of simple adaptors wrapped around existing systems.

Such a technological advance may enable novel applications also in the area of design optimization. Contemporary design optimization systems are usually confined within the boundary of a single organization. Modern engineering products, on the other hand, are assembled out of components developed by several organizations. One may devise a scenario in which each organization makes it available on the Internet services for simulating its own components and, based on such services, an inter-organizational service that simulates the behavior of the whole product is constructed. Such a composite service could then be used by a design optimization system. This approach could lead to more accurate results, simplify the tailoring of generic designs to specific scenarios, allow identifying fundamental design problems earlier, make it easier the management of product upgrades and so on.

Motivated by the above considerations, in this paper we explore such novel applications of simulation by focussing on some of its fundamental performance-related issues. A system involving several remote organizations can be managed in a centralized way, by an engine that moves data back and forth among all modules involved, or a in a distributed way, with data flowing from one module to the next. The former is much simpler to implement, but the latter may be much more efficient depending on the bandwidth available and the size of the data, for example. Gaining insight in this area is crucial for understanding how to structure design optimization systems spanning multiple organizations. There are also fundamental questions to be answered regarding the desirable trade-offs. For example, from the point of view of the system as a whole, one would like to maximize throughput, i.e., number of optimizations performed per unit of time. From the point of view of a participant organization, however, one would like to minimize the execution time of its services. These points of view may conflict because such services are usually associated with expensive software licenses that cannot be held by more than one process at a time. It follows that overlapping computation with communication at a node, a simple strategy for improving throughput, may lead to longer times in which a license is kept busy.

In this paper we evaluate by simulation the performance of an inter-organizational design optimization system in a variety of scenarios. We consider a system managed by a centralized engine and one managed by a distributed engine, in the above meaning. In each case, we study the behavior of a *block optimization scheduler*, i.e., one injecting a batch of candidate designs when the previous batch has been completed, and that of a *steady* optimization scheduler, i.e., one injecting a new design as soon as evaluation of the previous one has completed. Moreover, we study all these scenarios under two different patterns for data delivery, one in which a service starts only when all input data have been downloaded and one in which a service starts as soon as the first chunk of input data is available. Our analysis provides useful insights into the numerous trade-offs involved in the implementation of such a complex scenario.

2 APPLICATION DOMAIN

In this work we are concerned with *design optimization* systems. In this context optimization is an iterative process where each step is divided in two phases: the first phase is the evaluation of a set of proposed designs through simulation, the second phase is the generation of a new set of hopefully better designs. Each design represents a virtual prototype of an engineering product.

The strategy for producing new designs based on the evaluations of prior attempts is irrelevant to this discussion, e.g., Miettinen (1998), Quagliarella et al. (1997). The process terminates when either a satisfactory design is found or a predefined maximum number of designs have been evaluated. Evaluation of each design is performed according to a *simulation workflow* representing the sequence of tasks necessary for the analysis of the virtual prototype. As shown in Figure 1 design evaluation starts from the design variables x and leads to the performance measure $y = f(x)$.

A design optimization system is thus composed of two main blocks: the *optimization scheduler*, responsible for the synthesis on new designs from the prior ones; and the *workflow engine*, that receives evaluation requests from the scheduler and executes the simulation workflow to provide the corresponding results.

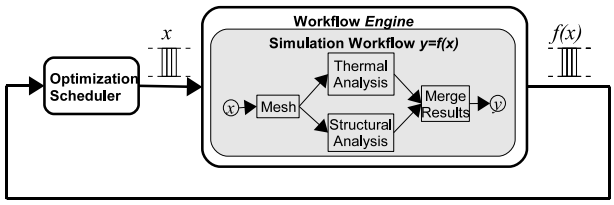

Figure 1: Simulation Based Optimization

2.1 Scenario

A workflow is: "*The automation of a business process, in whole or part, during which documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules*" (Workflow Management Coalition 1999). Simulation workflows adhere to this definition although they represent engineering simulations rather than business processes. Each workflow is defined by a workflow process definition (Aalst 1998) also called *schema*. A schema specifies which application programs

need to be executed, in what order and on which data. Each application program appears as a *service* that can be invoked by its clients (Papazoglou and Georgakopoulos 2003). A *workflow instance* consists of an execution of a schema on a specified set of input data and it produces a set of output data.

Execution of workflow instances is performed by a *workflow engine*, that interprets a high-level description of a schema. Description of workflow schemes and their execution is usually performed under control of a workflow management system (WfMS). The optimization scheduler, *scheduler* for short, is the client application for the WfMS. The scheduler requires design evaluations and waits for the simulation results. Then, it schedules new design evaluations according to the specific optimization algorithm used.

The gray box showed in Figure 1 depicts a simple simulation workflow within a design optimization system. The schema is composed of four services and describes the relationships among services as well as the data flow. This schema could be implemented as shown in Figure 2. Obviously, our notion of different organizations could correspond to different department or branches of the same organization. Communication across organizations occur, in general, through the Internet, which is the case of main interest in our work. In this example Organization C is the one interested in the optimization and invokes services provided by different organizations. Obviously, a given organization could provide more services and similar, or functionally equivalent, services could be provided by more organizations. The scheduler and the workflow engine could also communicate through the Internet, but we are not interested in emphasize this aspect.

The optimization scheduler is the closest process to the end users. An end user selects an optimization scheduler, feeds a simulation workflow to the optimization scheduler, and asks the scheduler to perform the optimization. In turn, the optimization scheduler submits workflow evaluations to the workflow management system.

Workflows are used in numerous fields of science and engineering, usually for elaborating data coming from experiments (Bose and Frew 2005, Johnston 2004, Ludäscher et al. 2005). Although we focus on workflows for design optimization systems, much of our analysis may be applied also to such contexts, in particular, whenever the delay incurred for transferring data between services is comparable to the computation time spent in services (see Section 7).

3 MODEL DESIGN

At the core of the simulation workflow there is a set of solvers. A *solver* is an application code specialized in a specific computation, e.g., finite elements of fluidodynamic analysis, and often it is a commercial licensed software. A solver execution usually proceeds as follows: it obtains

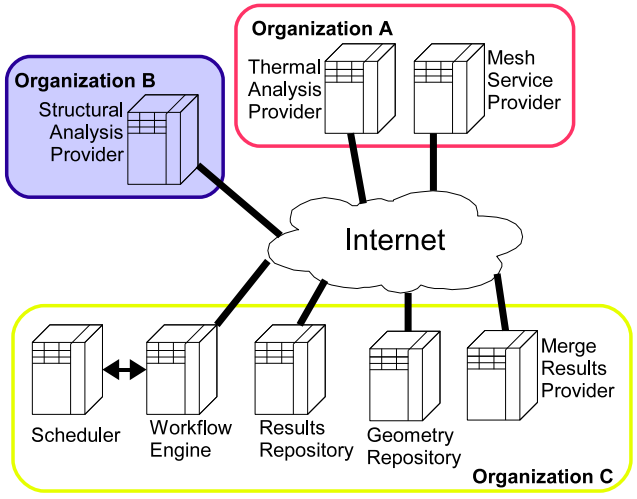

Figure 2: Physical Layout

an available license (the maximum number of solver instances that can be run simultaneously is bounded by the license agreement), reads the input data, performs the specific computation without any interaction with other entities, in particular, without interacting with other solvers, and finally it produces the corresponding output data. Input and output take the form of files.

The solver is wrapped by an ancillary code that we call *service* and takes care of all the communication activities required by the workflow engine, including the exchange of input and output files. Each service runs on a server. In most real-world settings, each server is tied to a specific solver.

3.1 The Service Structure

Data is exchanged through a pull protocol, i.e., each service is responsible for downloading the data it needs. Of course, the workflow engine must notify services about the location of the respective input files.

We analyzed the performance of two different interaction patterns among the components of the design optimization system. The first pattern, that we call *NoPipe*, works as shown in Figure 3: 1) the engine invokes a service, passing the relevant data links to it; 2) the service starts the downloading of the specified data; 3) when downloading has completed, the service starts the solver; 4) when the solver has completed, the service communicates the solver end to the engine. Only at this point the engine can start other services that depend on the data produced by the one just completed.

The second pattern, that we call *FullPipe*, works as shown in Figure 4: 1) as soon as there is output data available

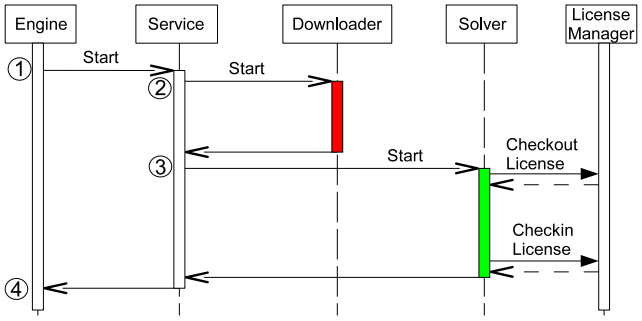

Figure 3: NoPipe Service Lifecycle

the service notifies the engine; 2) the engine, according to the workflow schema invokes the subsequent services and a data pipeline is established between pairs of services; 3) each service starts both the downloader and the solver. Each solver obtains a license as soon as it starts, like in the previous case; we did not include the corresponding actions in Figure 4 for the sake of clarity. The service notifies the engine as soon as there is some output data available for downloading (this step is not shown in the Figure 4). Thanks to elaboration and transmission overlap, we expect this pattern to provide better performance in terms of overall throughput, but we also expect it to give worse results in terms of resource (license) usage.

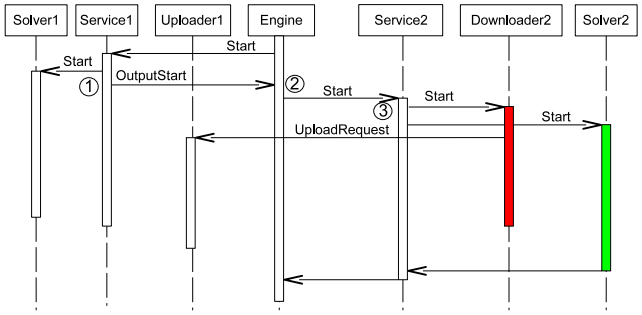

Figure 4: FullPipe Service Lifecycle

3.2 Data Storage

A design optimization system is usually implemented in a *centralized* way, as follows. There is a single repository that can be accessed by all modules (i.e., workflow engine, optimization scheduler, services). The workflow engine stores input and output data generated by the services in this repository. Each service reads input data from the repository, stores them on local storage and writes the final output data back to the repository.

When the services are distributed geographically, as in a multi-organizational system, it makes sense to explore a *distributed* storage architecture, in which the output data of a service are passed directly to the service that needs that data, i.e., without moving such intermediate results back and forth to the workflow engine. In this case, the repository at the workflow engine stores only the final results produced by each instance. The two alternatives are depicted in Figure 5.

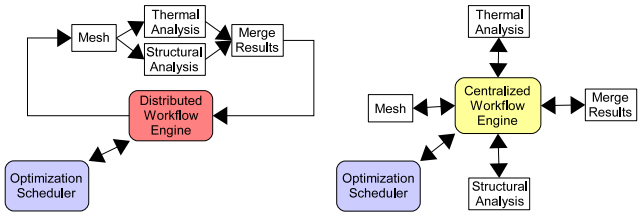

Figure 5: Storage Architectures

3.3 The Scheduler Policies

The optimization scheduler may submit new tentative designs for evaluation according to several different policies. We explored the two policies most commonly used, as follows. With the *block* submission policy the scheduler first generates a batch of n workflow instances (designs), submits all these instances to the workflow engine and waits for completion of all of them. The scheduler then generates a new batch based on the results of the previous batch, and so on. This submission pattern is typical of optimization schedulers based on generations of population such as genetic algorithms and evolutionary strategies (Carson and Maria 1997). With the *steady* (or *steady-state*) policy the scheduler starts a batch of n instances and then submits a new evaluation as soon as a previous one has completed. Note that submission policies are largely independent of the specific search strategy implemented by the scheduler.

4 MODEL IMPLEMENTATION

We simulated the scenario of our interest using DESMO–J (2000), a discrete event simulation framework written in Java. The scenario was modelled as follows.

We defined the following processes: optimization scheduler, workflow engine, service instances, uploaders, downloaders and solvers. The entities we modeled are: hosts and the file system components. Finally the resources we modeled are: link connections, file systems, disks and the licenses available for running the solvers.

Hosts are the sites providing services, they have: unlimited memory, a number of CPUs at least equal to the number of licenses for the solvers (this is common in practice) and a file system shared among all the processes running in the host. Solvers elaborate input data at a rate specified by their elaboration speed.

The file is the basic entity needed to model the host file system. The file is modeled as a list of disk sector. The file system is the intermediate between the host processes and the host disk, it contains a fixed size buffer cache based on a Last Recently Used buffer replacement policy. The processes access local data via files, because files are part of the file system every read or write request is mediated by the file system and the buffer cache. File system metadata is not stored on the disk. Disks are modeled as unlimited lists of sector, read and write operations are queued in the same fifo queue. Disks are characterized by the sector size and by the minimum and maximum time for read and write a single sector.

There is a communication link between each pair of hosts. Links are characterized by a mean latency time and a mean bandwidth. Links have also a dynamic parameter: the number of active connections, an active connection is a connection that is feeding data into the link. The bandwidth of a link is fairly distributed among all the active connections. Processes create connections over a specified link in respond to a file transfer request, then they write and read data to and from the connection a chunk at time. A chunk is a unit of data with a size equal to the disk sector size.

5 PERFORMANCE METRICS

We define as *optimization latency* the time required for completing the optimization session, we define as *throughput* the number of evaluated workflow instances divided by the optimization latency. We define as *solver latency* the time required for one solver execution. This time includes reading of input data from the local disk, their processing and the writing of output data on the local disk. We define as *service latency* (or service *lifespan*) the time required for one service execution. This time includes the downloading of input data, the solver latency, the uploading of output data.

A key element of our analysis is the cost incurred by each organization participating in the workflow. We assume the *service cost* is proportional to the service lifespan, whereas the *solver cost* is proportional to the solver latency — recall that a solver performs a license checkin as soon as it starts and the corresponding license checkout when it is about to finish. These definitions of the service cost and of the solver cost are associated with one workflow instance. The cost associated with the complete workflow is simply the sum of the cost associated with all the instances.

6 EXPERIMENTS AND RESULTS

We conducted experiments on a simple workflow schema, composed of four identical services serially connected as shown in Figure 6. We assume that at each stage the size of the output data is equal to the product of the size of input data for the output/input ratio, each service behaves as described in Section 3.1.

We performed a number of experiments around a working point described by the parameter values in Table 1. The buffer cache size is expressed in number of disk sectors. We modeled the disk access times as random variables with uniform distribution. We modeled link delay and bandwidth as random variables with gaussian distribution, with standard deviation equal to $1/10$ of the mean value.

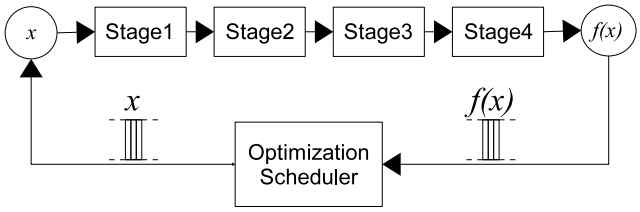

Figure 6: Serial Workflow Schema

Table 1: Working Point

Parameter	value
Buffer Cache Size	100000
Minimum Disk Read Time [s]	0.001
Maximum Disk Read Time [s]	0.002
Minimum Disk Write Time [s]	0.0002
Maximum Disk Write Time [s]	0.0004
Sector Size [B]	8192
Initial Data Size [B]	1000000
Batch Size	10
StartUp Time [s]	30
Elaboration Speed [B/s]	65536
Output/Input Ratio	1.1
Mean Link Delay [s]	0.001
Number of Batches	20

All the experiments have been conducted with both storage architectures (centralized vs. distributed), with both scheduler policies (block vs. steady) and with both data delivery patterns (NoPipe vs. FullPipe). This analysis has enabled us to gain insights into the relationships between the numerous systems available. We remark, however, that the modules used in contemporary design optimization systems may restrict such choices. In particular, some optimization algorithms are inherently steady while others are inherently block based. Moreover, the majority of solvers do not support the FullPipe data delivery pattern.

The following three subsections show the obtained results. The first is related to the NoPipe pattern, the second is related to the FullPipe pattern and the third describes the results in an aggregated way. Each plotted value on the charts is the mean of 20 simulation runs.

6.1 Results for NoPipe

Figure 7 shows throughput as a function of link bandwidth (note the logarithmic scale on the Y-axis). For small values, bandwidth is the system bottleneck then for larger values system throughput becomes independent from the bandwidth, the storage architecture and the scheduler policy. The steady policy achieves greater throughput than the block policy, the distributed storage achieves greater throughput than the centralized one.

It can be seen that when the link bandwidth is the system bottleneck, a Steady scheduler achieves better throughput than a Block scheduler. The improvement is about 200% for bandwidth in the range 10000-100000 B/sec, which may be representative of the bandwidth available in a WAN environment. As expected, when the bandwidth is no longer an issue, throughput is limited by other bottlenecks and becomes independent of both the storage architecture and scheduler policy.

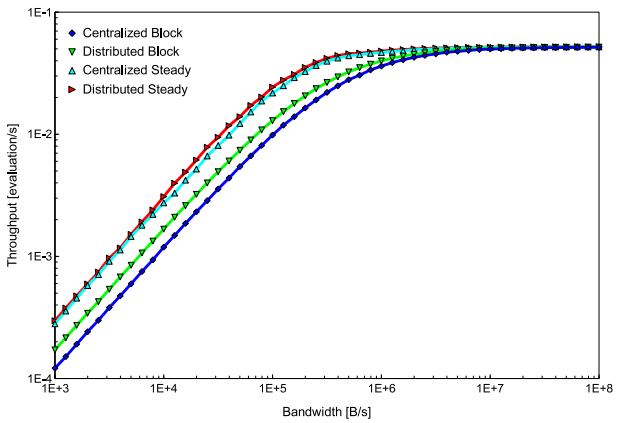

Figure 7: NoPipe Throughput

Figure 8 describes the effect of the storage architecture. It shows the ratio of throughput with distributed storage to throughput with centralized storage. It can be seen that a distributed storage is always beneficial for a block scheduler, whereas for a steady scheduler the difference in the two cases is less sensible. Obviously, when the link bandwidth is no longer a bottleneck, whether the architecture is distributed or centralized becomes irrelevant from the point of view of throughput (ratio tends to 1).

Figure 9 shows the service lifespan measured on the fourth stage of the workflow schema versus the link bandwidth. It can be seen that the centralized storage guarantees a lower lifespan (note that the Y-axis has a logarithmic scale). The improvement over the distributed storage is about 100% for bandwidth in the range 10000-100000 B/sec, which may be representative of the bandwidth available in a WAN environment.

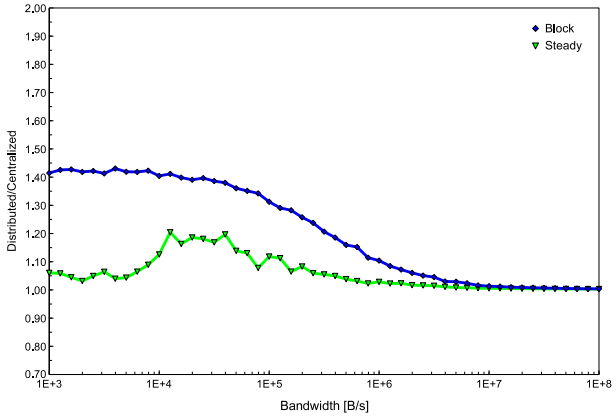

Figure 8: NoPipe Throughput Ratio (Distributed/Centralized)

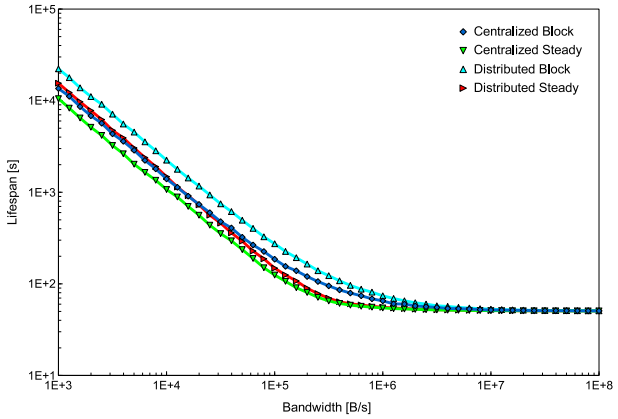

Figure 9: NoPipe Service Lifespan

Figure 10 shows the effect of the storage architecture, in terms of ratio of service lifespan in the distributed case vs. service lifespan in the centralized case. Once again, distribution is more beneficial for a block scheduler especially when the available bandwidth is small. The solver latency is not shown because with the NoPipe pattern it is essentially independent of the link bandwidth.

6.2 Results for FullPipe

Figure 11 shows the throughput versus the link bandwidth, almost all curves are superimposed (on a logarithmic scale) within an envelope less than 10%. Figure 12 shows the throughput speedup and it confirms that the maximum throughput difference between centralized and distributed storage is around 10%. Figure 13 shows the service lifes-

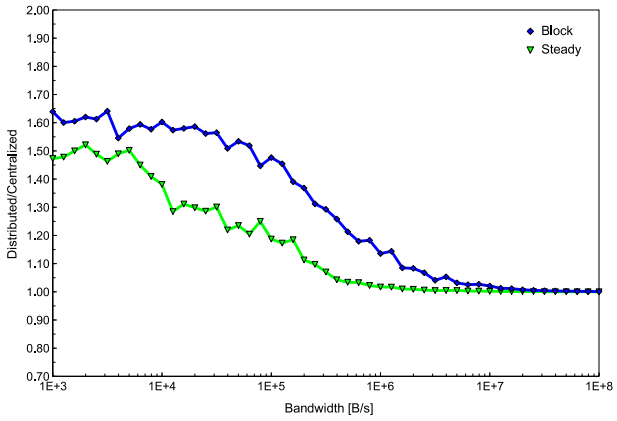

Figure 10: NoPipe Service Lifespan Ratio (Distributed/Centralized)

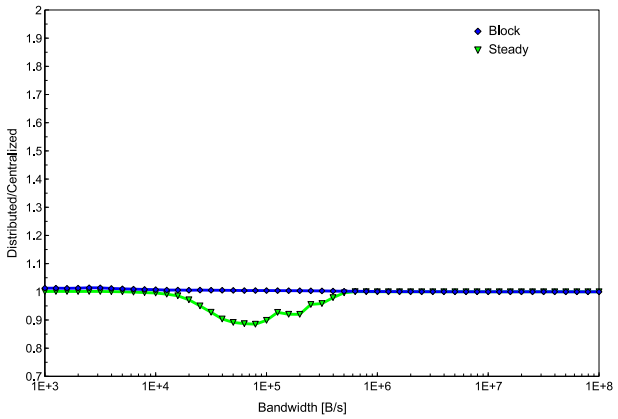

Figure 12: FullPipe Throughput Ratio (Distributed/Centralized)

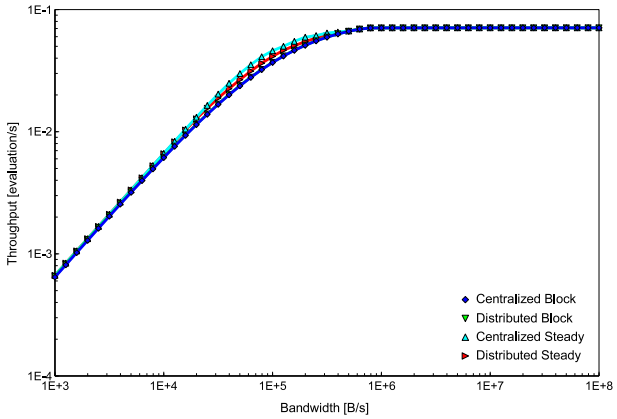

Figure 11: FullPipe Throughput

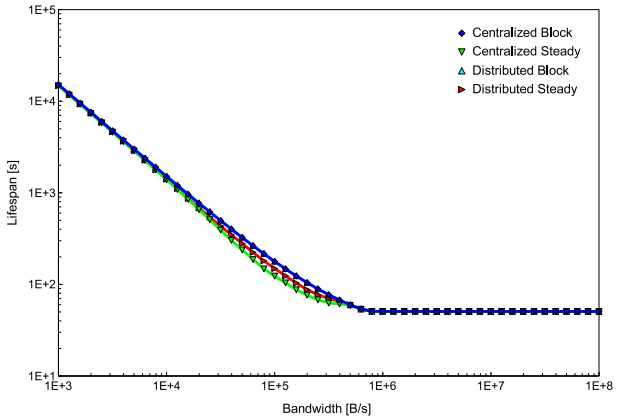

Figure 13: FullPipe Service Lifespan

pan versus the link bandwidth. Figure 14 shows the latency ratio of the service lifespan measured with the distributed and the centralized storage. With the considered bandwidth range and with the FullPipe pattern, throughput and service lifespan are practically unaffected by the scheduler and the storage architecture.

Figure 15 shows the solver latency versus the link bandwidth. The solver latency with the block scheduler is greater than the solver latency with the steady scheduler, and the solver latency with the distributed storage is greater than solver latency with the centralized storage. Note that Figure 15 also includes the solver latency for the NoPipe pattern (this value is constant for the entire bandwidth range). Figure 16 shows the solver latency ratio of the solver latency measured with the distributed and the centralized storage. For low bandwidth values distributed storage with the steady scheduler experiences a slowdown ranging from about 20% to about 90%.

6.3 Aggregated Results

Figure 17 shows the ratio of throughput with the FullPipe pattern to the NoPipe pattern. This figure says to us that, as long as the bandwidth is a bottleneck, we could have a significant throughput improvement when connecting the services in a data pipeline. On the other hand Figure 18 shows that the data delivery pattern has not a great impact on the service lifespan, the ratio of FullPipe to NoPipe pattern is never greater than 1.5 and in certain conditions it is lower than 1. Under such condition the FullPipe pattern is preferable also from the point of view of the service cost.

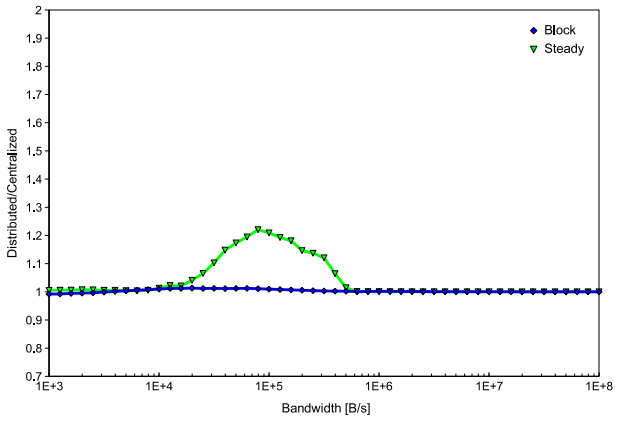

Figure 14: FullPipe Service Lifespan Ratio (Distributed/Centralized)

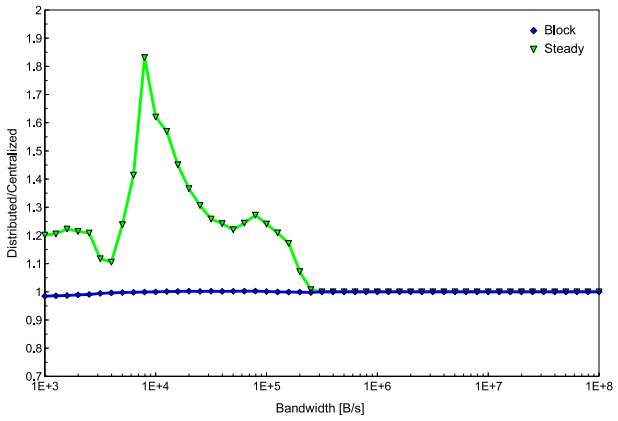

Figure 16: FullPipe Solver Latency Ratio (Distributed/Centralized)

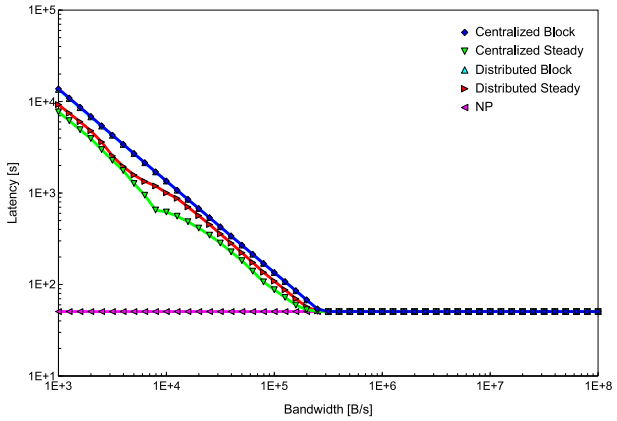

Figure 15: FullPipe Solver Latency

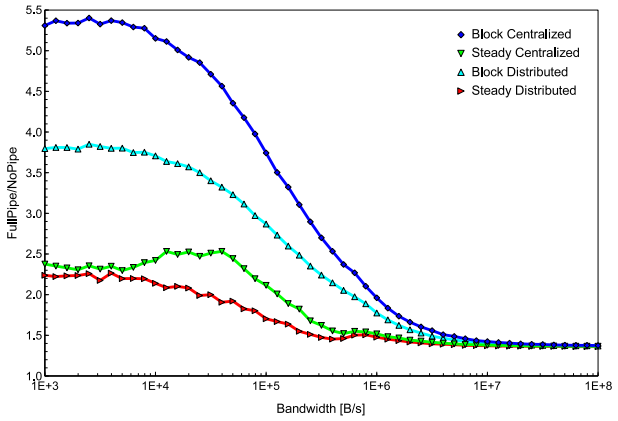

Figure 17: Throughput Ratio (FullPipe/NoPipe)

7 DISCUSSION

A first significant finding is that when link bandwidth is an issue storage architecture, scheduler policy, data delivery may all have a significant impact on performance. Although this result is not completely unexpected, it was not obvious whether the performance implications of the various implementation options are indeed significant. In particular, we found it difficult to predict the effect of the interactions between them. According to our analysis, on the other hand, it seems evident that when one moves to a bandwidth-limited environment, the previously mentioned options do have a strong impact on performance and thus they have to be analyzed carefully.

Having said this, it would be useful to provide useful rules for selecting the implementation options most suitable for a given scenario. It is evident, though, that our results cannot be used for predicting exactly the performance of any possible scenario, because of the large numbers of parameters involved and, perhaps most importantly, because the behavior of the overall system may be largely dependent on the specific workflow schema used.

Having warned that our results cannot be generalized easily, the key advices that we can draw from our simulations follow. Recall from the introduction that throughput is the key performance index from the point of view of the user of the design optimization system, whereas service cost and solver cost are the indices most relevant for the organizations that cooperate in the implementation of the system.

- When $\frac{bandwidth}{elaboration\ speed} > batch\ size$, it basically does not matter how the system is implemented:

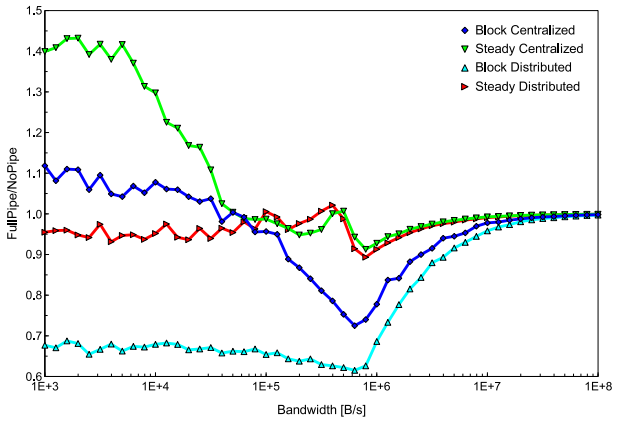

Figure 18: Service Lifespan Ratio (FullPipe/NoPipe)

all the options that we have analyzed exhibit more or less the same performance. When $\frac{bandwidth}{elaboration\ speed} < batch\ size$, on the other hand, different options may lead to strong differences in performance.

- To maximize system throughput one should use a FullPipe data delivery pattern. From this point of view, this strategy is always to be preferred to NoPipe even when bandwidth is not an issue (Figure 17). If FullPipe cannot be used (recall that many existing solvers do not support this option), then use of a steady scheduler is to be preferred over use of a block scheduler (Figure 7). In this case, whether storage is centralized or distributed is not very significant although distributed storage could provide some advantage.
- To minimize service cost one should use a centralized storage (Figure 9 and Figure 13). This conclusion is particularly useful, because a centralized storage is far simpler to develop, deploy and manage than a distributed one. A steady scheduler is to be preferred over a continuous one (Figure 10 and Figure 14). The choice between NoPipe and FullPipe is not univocally determined (Figure 18).
- To minimize solver cost one should use a NoPipe data delivery pattern. We note, also in this case, that NoPipe is much simpler to implement than FullPipe and, in particular, directly supported by most existing solvers. Alternatively we can obtain some minor improvements using a steady scheduler (Figure 15).
- The steady scheduler is almost always preferable to the block scheduler. Many generation based optimization schedulers have variations to run in a steady–state way (Vavak and Fogarty 1996).

ACKNOWLEDGMENTS

This work has been supported by ESTECO (<http://www.esteco.com>). Carlo Poloni, Luka Onesti, Cyril Fillon, and Eric Medvet provided useful comments at several stages of this work.

REFERENCES

Aalst, W. 1998. The application of Petri nets to workflow management. *The Journal of Circuits, Systems and Computers* 8 (1): 21–66.

Alonso, G., F. Casati, H. Kuno, and V. Machiraju. 2004. *Web services: Concepts, architectures and applications*. Springer Verlag.

April, J., F. Glover, J. P. Kelly, and M. Laguna. 2003. Simulation-based optimization: practical introduction to simulation optimization. In *Proceedings of the Winter Simulation Conference*, 71–78.

Banks, J., J. S. Carson, B. L. Nelson, and D. M. Nicol. 2005. *Discrete–event system simulation*. 4th ed. Upper Sadder River, New Jersey: Prentice–Hall.

Bose, R., and J. Frew. 2005. Lineage retrieval for scientific data processing: a survey. *ACM Computing Surveys* 37 (1): 1–28.

Botros, K., D. Sennhauser, K. Jungowski, G. Poissant, H. Golshan, and J. Stoffregen. 2004. Effects of dynamic penalty parameters on the convergence of moga in optimization of a large gas pipeline network. In *10th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*.

Boulougouris, E. K., A. D. Papanikolaou, and G. Zaraphonitis. 2004. Optimization of arrangements of ro-ro passenger ships with genetic algorithms. *Ship Technology Research* 51 (3): 99–105.

Carson, Y., and A. Maria. 1997. Simulation optimization: methods and applications. In *Proceedings of the Winter Simulation Conference*, 118–126.

DESMO–J. 2000. The desmo–j homepage. <http://www.desmoj.de>.

Fu, Y., B. Kachnowski, and E. Lee. 2004. Occupant model correlation using a multiobjective evolution strategy. In *10th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, Albany, New York, August 30-31.

Gaiddon, A., D. D. Knight, and C. Poloni. 2004. Multicriteria design optimization of a supersonic inlet based upon global missile performance. *Journal of Propulsion and Power* 20 (3): 542–558.

Giassi, A., F. Bennis, and J. J. Maisonneuve. 2004. Multidisciplinary design optimisation and robust design approaches applied to concurrent design. *Structural and Multidisciplinary Optimization* 28 (5): 356–371.

Gottschalk, K., S. Graham, H. Kreger, and J. Snell. 2002. Introduction to web services architecture. *IBM Systems Journal* 41 (2): 170–177.

Hepp, E., O. Lohne, and S. Sannes. 2003, November. Extended casting simulation for improved magnesium die casting. In *DGM 6th International Conference on Magnesium Alloys and Their Applications*, 669–674. Wolfsburg, Germany.

Johnston, W. E. 2004. Semantic services for grid-based, large-scale science. *IEEE Intelligent Systems* 19 (1): 34–39.

Ludäscher, B., I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. Jones, E. A. Lee, J. Tao, and Y. Zhao. 2005. Scientific workflow management and the kepler system. *Concurrency & Computation: Practice & Experience*. To appear.

Maisonneuve, J. J., S. Harries, J. Marzi, H. C. Raven, U. Viviani, and H. Piippo. 2003. Towards optimal design of ship hull shapes. In *Proceedings of the 8th International Marine Design Conference*, 31–42.

Miettinen, K. M. 1998. *Nonlinear multiobjective optimization*, Volume 12 of *International Series in Operations Research & Management Science*. Springer Verlag.

Papazoglou, M. P., and D. Georgakopoulos. 2003. Service–oriented computing: Introduction. *Communications of the ACM* 46 (10): 24–28.

Quagliarella, D., J. Périaux, C. Poloni, and G. Winter. (Eds.) 1997. *Genetic algorithms and evolution strategies in engineering and computer science*. West Sussex, England: John Wiley and Sons.

Stal, M. 2002. Web services: beyond component-based computing. *Communications of the ACM* 45 (10): 71–76.

Swisher, J., P. Hyden, S. Jacobson, and L. Scruben. 2000. A survey of simulation optimization techniques and procedures. In *Proceedings of the 2000 Winter Simulation Conference*.

Taga, I., A. Funakubo, and Y. Fukui. 2005. Design and development of an artificial implantable lung using multiobjective genetic algorithm: Evaluation of gas exchange performance. *ASAIO Journal* 51 (1): 92–102.

Vavak, F., and T. Fogarty. 1996. Comparison of steady state and generational genetic algorithms for use in nonstationary environments. In *Proceedings of the 1996 IEEE Conference on Evolutionary Computation*: IEEE Press.

Workflow Management Coalition 1999. *Workflow management coalition terminology & glossary*. Workflow Management Coalition. Document No. WFMC-TC-1011.3.

AUTHOR BIOGRAPHIES

PAOLO VERCESI is a PhD student in Computer Science at the University of Trieste, Italy. He is an Information Technology consultant for ESTECO. He is a member of the IEEE Computer Society. His research interests include distributed computing and business process modeling. His e-mail address is <pvercesi@univ.trieste.it>.

ALBERTO BARTOLI is an associate professor of Computer Engineering at the University of Trieste, Italy. His research interests are in the area of reliability in distributed computing His Web address is <http://www.univ.trieste.it/bartolia>.