Saleve: Simple Web-Services Based Environment for Parameter Study Applications

Zsolt Molnár Imre Szeberényi

Abstract—**The goal of the Saleve Project is to develop and evaluate mechanisms and abstractions that may connect the diverse research community of the distributed (mainly the Grid) computing to those users, who are not familiar with distributed computing as such, but who would simply like to use the results in their everyday tasks. We show a simple web-services based, domain-specific computational framework that integrates smoothly into the well-known, traditional user environments and requires learning no new technologies.**

***Index Terms*—Parameter Study Applications, Web Services, Computational Grid**

I. Introduction

The distributed computing paradigm, and the researches carried out to exploit its benefits have become one of the most exciting and supported computing challenges in the past decade. The prolific, not compatible, different tools, methods and approaches demanded to invent and create common platforms and standards. However, those "standards" themselves have been subject to rapid development and changes. The tools implementing them are also changing frequently; therefore users must adapt the applications and the environment built on top of them with almost the same frequency. It is generally a heavy burden. Saleve [1] intends to ease the situation by hiding the technological details from the end-user, and provides a generic, lightweight, instantly usable solution for the well-specified problem domain of the *parameter study (*or *sweep* - PS*)* tasks on top of the present distributed technologies. Roughly, a PS task involves a domain (*parameter space*) and an *operation* that "sweeps" over the domain to produce the desired result. The domain may be divided into sub-domains and the operation can be executed independently on them, that yields the base of the distribution of the computation. PS tasks cover a wide range of technical and scientific research activities. Extremely complicated PS tasks are involved in high energy physics, astronomy, cosmology data analysis, data aggregation, etc. Fortunately, the PS tasks share a simple, common structure in the distributed computing context.

The most emerging distributed computing paradigm, the Grid [2] concept, introduced the Grid portals as the central entry points for the Grid services. They are mostly web based (like GridSphere [3]); the portal user can create new services in well-defined ways, submit and monitor jobs, etc. However, this kind of environment usually cannot interact smoothly with the older, well-tested tools. Saleve attempts to bridge this gap as well.

First of all, let us review the relationship to some related projects emphasizing the major differences only. The most similar project is AppLeS [4]. AppLeS also intends to simplify and automate the PS task distribution. While AppLeS focuses on its own scheduling methods, Saleve relies completely on third-party schedulers (that are completely hidden from the end-user). The second main difference is the communication method: AppLeS uses ssh, Saleve uses DIME-enabled [5] web services to ease the smooth integration into the grid portal settings. The third main difference can be observed in the account and credential handling. Using the ssh method, the user must have real, personal account on the server machine that makes group-wise activities and access sharing difficult. Saleve approaches the problem by providing a virtual "user" entity to the computing resource that may be shared further among the real clients.

The following projects may be regarded as one group: Nimrod [6], Condor [7], ClusterGrid [8], BOINC [9], and Globus [10]. Saleve is built on top of them; in fact they represent those third party schedulers. Generally, the user must create explicit task descriptor files, and/or either a special toolset must be available on the user's desktop or the user must log in directly to the executing resource. Saleve eliminates both requirements: the task description constitutes a natural part of the user's application program, therefore it is written in the user's convenient and well-known language. The executing "tool" is also part of the program, the application knows how and where to execute itself. No special client-side runtime environment is needed.

We expect that our approach speeds up the PS application development and execution, and deepens the cooperation

Manuscript received June 6, 2005. This work was supported by NKFP grant 2/009/2004, IKTA grant 00064/2003 Hungarian and INFSO-50883 (EGEE) EU funds.

Zs. M. Author is with the Budapest University of Technology and Economics, Tapiontie 6B 26, Oulu, 90570, Finland (phone: +358 40 702 0663; e-mail: zsolt.molnar@ieee.org).

I. Sz. Author is with the Budapest University of Technology and Economics, Budapest, 1111, Muegyetem rkp. 3-9, Hungary (e-mail: szebi@iit.bme.hu).

between the members of smaller and geographically distributed research groups.

II. Properties of Saleve

A. Structure and Behavior.

SALEVE has client-server architecture. The client provides the entire set of executable and data files, the server renders and directs them further to the underlying distributed system (Fig. 1.).

The client side workstation is the location of the application development. We assume that the developer is less familiar with distributed technologies. The developer creates the calculation by using the Saleve client library and his/her programming language constructs. Inside the program, the developer registers those resources that are required to execute the task. For the moment, the resources are input/output and executable files. Only the registered resources are transferred to the server, and back to the user. Then the parameter space partitioning must be executed. *The way of the partitioning is arbitrary*: it may be embedded into the application; it may be based on user-defined description files, etc. Optional client-side command line parameters may be defined as well, therefore Saleve has only small impact on the interface between the user and his/her own system.

After those modifications, the produced executable itself becomes the Saleve client. It is responsible for handling its own resources and implements automatically the required communication with the server.

A Saleve client may run in two, *local* or *remote* modes. In *local* mode, all parts of the calculation are executed on the launching node, no Saleve server is involved - the client behaves like a traditional application. But the Saleve client library provides a little more: If the local machine is a multiprocessor system, then it is able to launch more partial calculation instances at the same time! So a Saleve client application *is automatically a multiprocessor application*.

We introduced this mode in order to ease testing and application development. Only final products should be submitted to the computing servers; testing, optimization and fine-tuning should be performed locally. When the user runs the program in *remote* mode, the sequential execution of the partial calculations turns into (optimized) data transfers using *web services* technology. We refer to this method as *a virtually sequential execution* of a parallel task. The remote Saleve server receives the material, submits it to a cycle service, and transfers back the results.

To run the application remotely, the URL of a Saleve server must be provided. The URL may point to a real Saleve server or a dispatcher service giving the same Saleve interface like a real Saleve server. Based on the application, load, platform, etc. requirements, the task may be transferred to other Saleve servers by the dispatcher. The reference implementation provides such a transferring server.

On the local machine, the (virtual) calculation is over when all the partial results arrived (independently form their way of producing). The post-processing is then performed locally again. The distributed system is completely hidden behind the Saleve server: once a new technology appears, and an adapting Saleve server is set up, the user can run the calculation using that technology immediately, *without any modifications/recompilations* on his/her side.

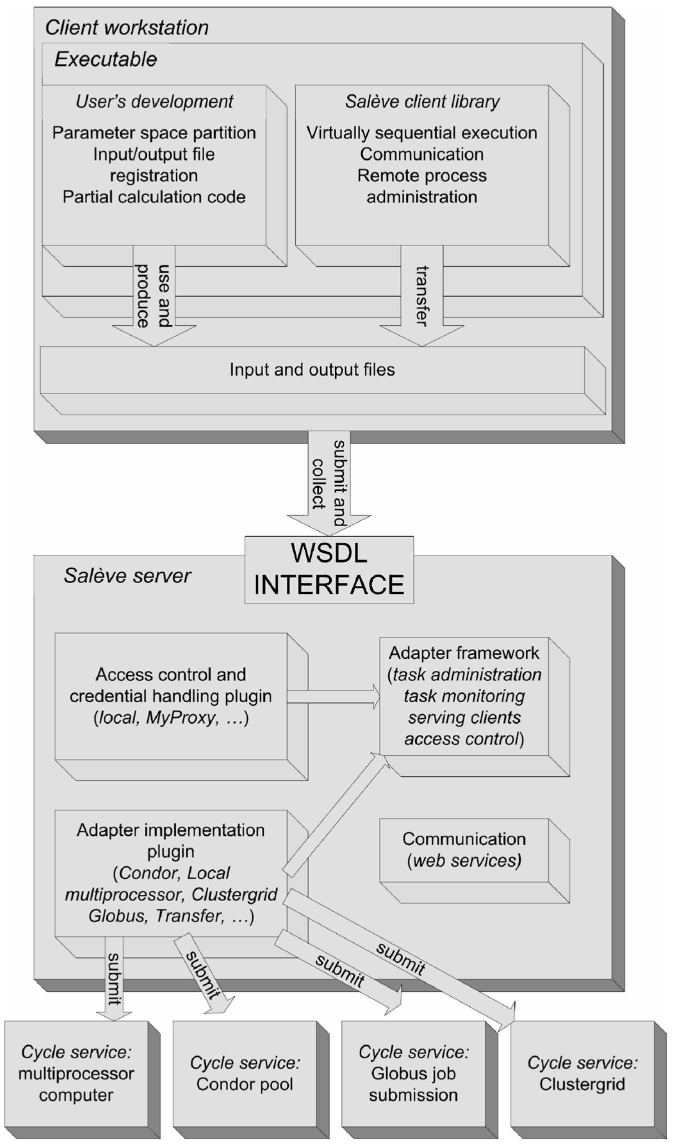
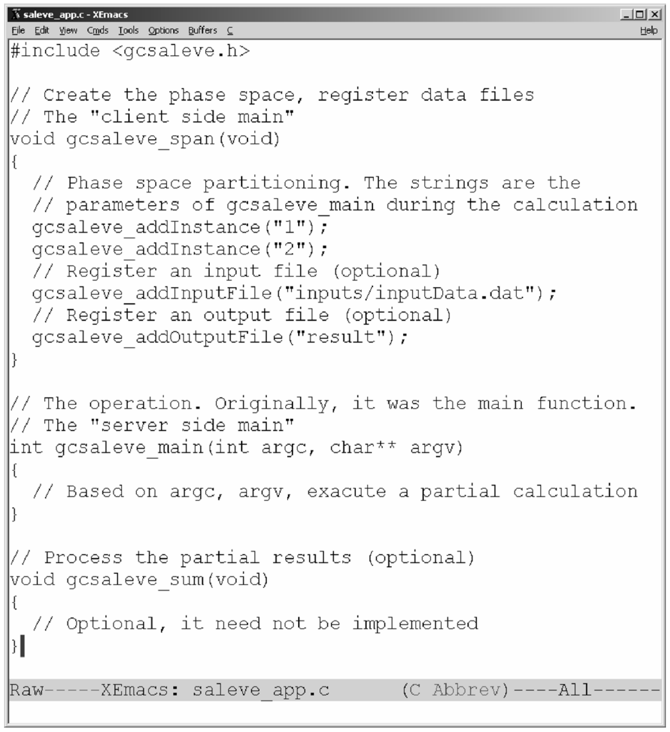

Client workstation
Executable
User's development
Parameter space partition
Input/output file registration
Partial calculation code
Salève client library
Virtually sequential execution
Communication
Remote process administration
use and produce
transfer
Input and output files
submit and collect
WSDL INTERFACE
Salève server
Access control and credential handling plugin (local, MyProxy, …)
Adapter framework (task administration task monitoring serving clients access control)
Adapter implementation plugin (Condor, Local multiprocessor, Clustergrid Globus, Transfer, …)
Communication (web services)
submit
submit
submit
submit
Cycle service: multiprocessor computer
Cycle service: Condor pool
Cycle service: Globus job submission
Cycle service: Clustergrid

Fig. 1. Structure of Saleve

All the client (application developer) side functionality can be accessed from application development languages. For the moment, C and C++ are supported, but with tools like f2c of g77 [11], FORTRAN may easily be used as well. It means that if the user can access and modify the source code of a traditional application, then he/she may easily turn it into a distributed one. Explicit task-distributing constructs (like computing nodes, job submitting, remote procedure calls, etc.) are hidden from the user. Fig. 2. contains a simple code snippet where all the Saleve C language constructs are shown up.



```
#include <gcsaleve.h>

// Create the phase space, register data files
// The "client side main"
void gcsaleve_span(void)
{
  // Phase space partitioning. The strings are the
  // parameters of gcsaleve_main during the calculation
  gcsaleve_addInstance("1");
  gcsaleve_addInstance("2");
  // Register an input file (optional)
  gcsaleve_addInputFile("inputs/inputData.dat");
  // Register an output file (optional)
  gcsaleve_addOutputFile("result");
}

// The operation. Originally, it was the main function.
// The "server side main"
int gcsaleve_main(int argc, char** argv)
{
  // Based on argc, argv, exacute a partial calculation
}

// Process the partial results (optional)
void gcsaleve_sum(void)
{
  // Optional, it need not be implemented
}
```

```
Raw-----XEmacs: saleve_app.c      (C Abbrev)----All------
```

Fig. 2. **Saleve C language constructs**

An important problem arises when the application would use some *shared libraries*. Saleve cannot ensure that the required libraries, versions, etc. be present on the executing nodes. Therefore, the shared code should also have to be submitted in a way. Saleve does not provide support for this part, but in most cases statically linked binaries work well. When working with Saleve, all the libraries should be linked statically to the executables.

B. Communication

The communication between the client and the server is implemented by using *the web services technology*, based on SOAP [12]. Selecting it has some obvious reasons: it is a *de facto* standard in the Grid service development community, it uses (optionally secure) HTTP protocol (that is supported by almost every networked computer), and it is supported by more and more stable and well-tested tools. It provides interoperability across institutional and application language boundaries; the client has to assume nothing about the implementation details of the server (and vice versa). The SOAP based communication in the reference implementation was developed by using gSOAP [13]. gSOAP does not require any pre-installed runtime environment. Using the WSDL [14] description of the Saleve service, it generates stub and skeleton codes in ANSI C/C++. Therefore, the whole communication subsystem is generated in source code level, and it is fully integrated into the client library. The major consequence is that *an arbitrarily implemented Saleve client can interact with an arbitrarily implemented Saleve server* by using only the basic and traditional HTTP communication without the burden of developing and maintaining any custom protocol and client-side runtime environment.

During a Saleve process, the reliable transfer of big binary files is mandatory. gSOAP provides stable DIME attachment handling, that supports even arrays or structures of binary files.

C. Fault tolerance.

In remote mode, networked communication occurs between the client application and the remote server. The networked communication frequently suffers from interruptions and other uncontrollable error situations. On the other hand, the mobility of the computers and networked devices is a demand today – causing unavoidable network services cut from time to time. The Grid portals would provide a solution for this problem as the interface and job execution is remote, only the handler device is local. Saleve must provide the same capability. After the communication subsystem finished the transferring of the job data (files, etc.), the execution switches to polling mode: it checks regularly the state of the job, and retrieves the ready partial results. If the file retrieval fails, then the system restarts it later. This mode can be interrupted (either by the user or by the network), and resumed later. Network problems are reported by the client, but the virtual execution does not stop in this case. Each Saleve client has proper and uniform command line interface to re-attach to an interrupted job, because *the state of a running job is controlled and tracked by the remote Saleve server*.

Another question is the case of the *server side fault*. If the underlying service fails, Saleve reports it to the client that may warn the user. If the Saleve server fails, the client interrupts the execution when timeout occurs. Time to time, the server saves the task state. By using this checkpoint, the server is able to resume the task, independently from the origin of failure. The user may re-attach the client to the task at any time, from even different locations. The task state and data is cleaned up only when task termination is explicitly called from the client.

D. Job and User Administration, Access Control

The basic authentication system of Saleve is the HTTP authentication. Based on this, the remote Saleve server either determines the access rights of the user itself, or maps it to the credential handling system of the underlying job manager (by using *authentication plugins*, see later). A Saleve client always asks for password before the first communication attempt with a remote Saleve server. Users may store their user name and password in encrypted files as well. The reason of choosing HTTP authentication is the fact that most front-end desktops support this. That seems to be sufficient for user identification. The centrally managed Saleve server uses more elaborate access controlling methods on behalf of the user, making certificates toward the real resources better protected.

Saleve provides client side tools for user management, job control and job monitoring as well. All the required functionalities are implemented in the Saleve client side library, so they are available by using each Saleve client. The user may run an arbitrary Saleve client program with command line parameters to execute an administration task. The user has his calculation and administration tools built in one executable making the work with Saleve extremely powerful and location independent

E. *Saleve Server Capabilities.*

The Saleve server also has multiple roles and tasks. First of all, it centralizes the management of the distributed system: if the system changes, only the Saleve server has to be changed, the applications can be used without any modification. It also represents a service consumer to the Grid (in case of the Grid based implementation). As the Saleve server is only one entity from the viewpoint of the Grid, only simple access policy has to be set up and maintained. The Grid administrators know that Saleve uses the resources only in a well defined way, for a well-defined task type – they could provide *better, optimized scheduling* (for instance by using AppLeS).

F. *Plugins.*

To adapt the new or changed technologies faster and more easily, the Saleve server project provides a framework implementing the most common functionalities. The technologies are linked to the server by plugins. *Adapter* (submission), *authentication* and *file handling* plugins must be provided.

G. *Reference implementation.*

The reference server was written in C++, by using gSOAP. It may work in two modes: *daemon* and *CGI* modes. In daemon mode, a listening port must be given as a parameter.

The reference implementation provides two authentication plugins. The first one provides basic services only: it stores encrypted user name/password pairs locally. This plugin is suitable for those distributed systems that have no intrinsic user management and access control capabilities. The second one is suitable for Grid systems. It allows Saleve to download Grid certificates (proxy credentials) from a MyProxy [15] server. The users of Saleve do not have to obtain a valid certificate, only the Saleve administrator has to do it once.

There are four adapter plugins ready. The first one is valuable for local multiprocessor (supercomputer) systems. This is the simplest case. There is no other distributed technology behind; it is the most simple, stand-alone scheduler, enabling remote job execution on a (super)computer. Its main purpose is to provide test or dummy Saleve servers. The supported production systems are the following: Condor, Globus, ClusterGrid. We are planning to develop more plugins (BOINC, gLite [16], BirdBath [17]) that may make the underlying systems directly comparable.

III. Selected Problems Solved with Saleve

SALEVE system was evaluated and tested by adapting and executing complicated high energy physics applications, like Event2 [18] and NLOJET++ [19]. We reproduced easily and effectively those results that were published in [20]. At the current state of the project we started to demonstrate the development simplicity and speed with a quite new application that makes global parameter scanning for solving boundary value problems using PHA algorithm [21].

IV. Grid Portals and the Future

AN important property of the Saleve client applications is that they can be turned into portal services immediately, by using traditional methods. The data of the parameter space distribution can be produced by simple portlets, designed specially for the application. The output of a Saleve client is always well-defined, therefore it can be directed to other portlets that may understand, visualize and analyze it further. The application with an URL pointing to a Saleve server constitutes one calculating entity with a well-defined task that knows where and how to execute itself. A Saleve client, created by a scientist not knowing much about the distributed technologies, becomes a powerful Grid portal service – that was the leading vision of the Saleve project.

References

[1] Saleve Project home page, http://gcsaleve.sourceforge.net

[2] I. Foster, C. Kesselman, J. M. Nick, S. Tuecke, ”The Physiology of the Grid An Open Grid Services Architecture for Distributed Systems Integration,” Open Grid Service Infrastructure WG, Global Grid Forum, June 22, 2002., http://www.globus.org/alliance/publications/papers/ogsa.pdf

[3] GridShere homepage, http://www.Gridsphere.org

[4] AppLeS Home Page, http://grail.sdsc.edu/projects/apst/

[5] Direct Internet Message Encapsulation (DIME), June 17, 2002., http://msdn.microsoft.com/library/en-us/dnglobspec/ html/draft-nielsen-dime-02.txt

[6] Nimrod Home Page, http://www.csse.monash.edu.au/~davida/nimrod.html/

[7] Condor Project Home Page, http://www.cs.wisc.edu/condor/

[8] The Hungarian ClusterGrid Infrastructure project, home page, http://www.clusterGrid.hu/

[9] BOINC Home Page, http://boinc.berkeley.edu/

[10] Globus Team, Globus Toolkit, http://www.globus.org

[11] g77 Home Page, http://world.std.com/~burley/g77.html

[12] SOAP Version 1.2, W3C Recommendation 24 June 2003., http://www.w3.org/TR/soap12

[13] gSOAP home page, http://gsoap2.sourceforge.net/

[14] Web Services Description Language (WSDL) 1.1, W3C Note, 15 March 2001., http://www.w3.org/TR/wsdl

[15] J. Novotny, S. Tuecke, V. Welch, “An Online Credential Repository for the Grid: MyProxy,” Proceedings of the 10th IEEE Intl. Symp. on High Performance Distributed Computing, 2001.

[16] gLite Home Page, http://glite.web.cern.ch/glite/

[17] BirdBath Home Page http://www.cs.wisc.edu/condor/birdbath/

[18] Home page of Event2, http://hepwww.rl.ac.uk/theory/seymour/nlo/

[19] Home page of NLOJET++, http://www.cpt.dur.ac.uk/~nagyz/nlo++-v2/

[20] Z. Nagy, Z. Trocsanyi, “Multi-jet cross sections in deep inelastic scattering at next-to-leading order,” Phys.Rev.Lett. 87 (2001), 082001

[21] G.Domokos, I.Szeberényi, ”A hybrid parallel approach to one-parameter nonlinear boundary value problems,” Computer Assisted Mechanics and Engieneering Sciences, 11:15-34, 2004.