

Bayanihan Computing .NET: Grid Computing with XML Web Services

Luis F. G. Sarmenta *

Sandra Jean V. Chua, Paul Echevarria, Jose Mari Mendoza, Rene-Russelle Santos, Stanley Tan,
Ateneo de Manila University, Loyola Heights, Quezon City, Philippines
and Richard P. Lozada
Microsoft Philippines, Makati City, Philippines

1 Introduction

XML web services are a new technology that promises greater ease-of-use and interoperability than previous distributed computing technologies such as DCOM, CORBA, and RMI, through the use of industry-standard XML protocols such as SOAP, WSDL, and UDDI [1]. While key industry players such as Microsoft, IBM, and Sun are already aggressively promoting XML web services as a way to improve business systems, we propose and demonstrate a new idea: that of using XML web services not only for business systems but for *grid computing* systems as well. In this paper, we present Bayanihan Computing .NET, a generic grid computing framework based on Microsoft .NET [1] that uses web services to: (1) harness computing resources through *volunteer computing*, and (2) make these resources easily accessible through easy-to-use and interoperable *computational web services*. In doing so, we achieve the two most fundamental goals in grid computing, and demonstrate the great potential of using web services for grid computing.

2 Volunteer Computing

The idea behind *volunteer computing* [2] is to allow ordinary users on the Internet to volunteer their idle computers' processing power towards solving computationally intensive tasks. Highly popular projects such as SETI@home and others have shown how volunteer computing can lead to supercomputer-like performance at very low costs. Bayanihan Computing .NET implements volunteer computing by providing a *PoolService* web service as shown in Fig. 1. This web service allows *computational clients* to create pools of tasks to be computed, and *volunteers* (or *workers*) to retrieve these tasks, perform them, and return their results. The *PoolService* is generic – that is, authorized computational clients can upload code for different applications in the form of an *assembly* (i.e., a DLL file), which is then automatically

*For more information, email lfgs @ alum.mit.edu or visit <http://bayanihancomputing.net/>

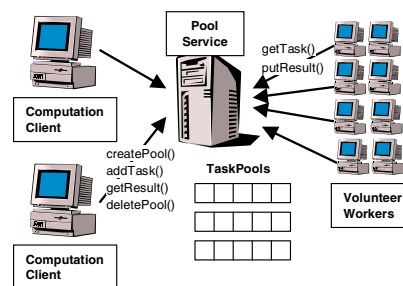


Figure 1. Volunteer computing with web services.

downloaded by the workers as necessary. (Security mechanisms in Microsoft's .NET framework allow the worker machines to safely execute these assemblies.) Thus, unlike application-specific systems such as SETI@home, Bayanihan Computing .NET can be used for different applications. In fact, the same *PoolService* can be used by different clients at the same time, as shown in Fig. 1.

3 Computational Web Services

The idea behind *computational web services* is to: (1) offer simple web methods that computational clients can call to perform application-specific computations on their own data, and then (2) use a parallel computing resource *behind-the-scenes* to perform the computation much faster than possible on a single machine. Figure 2 shows an example of how a computational web service can hide a volunteer computing system. Here, the computational client does not connect directly to the *PoolService*, but instead connects to a *MandelService* web service. In this way, we shield the client's programmer from the complexity of parallelizing the rendering task. As shown in Fig. 3, all the client has to do is call the *Render()* web method and receive the finished bitmap. Behind-the-scenes, it is the *MandelService* that parallelizes the computation and lets it run on the volunteer computing system by communicating with the *PoolService* accordingly.

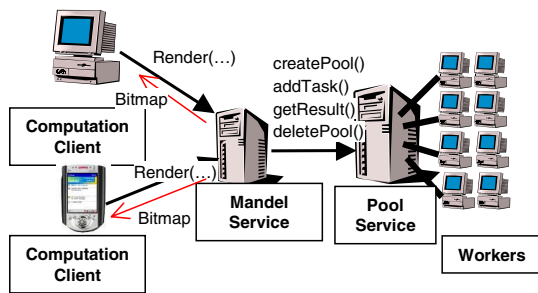


Figure 2. Computational web services hide parallel computing resources behind easy-to-use web services.

```

MandelWebService mandelService
    = new MandelWebService();
byte[] [] result
    = mandelService.Render(
        2, 2, -2, -2, 400, 400, 2048 );
displayResult( result );

```

Figure 3. C# code for using the Mandelbrot web service.

In addition to making programming easier, computational web services also make it possible to have very “thin” clients. In fact, any device that can access web services – including handheld PDAs, as shown in Fig. 2, and potentially even cellular phones – can use these computational web services. In short, computational web services can, quite literally, bring supercomputing power to the *hands* of ordinary users.

Furthermore, computational web services are not limited to volunteer computing systems. With the appropriate backends, they can hide *any* high-performance processing resource in general – whether it be a supercomputer, a cluster, or even the Grid as a whole (including other computational web services), as shown in Fig. 4. By providing a simple interface to the programmer, computational web services make it possible to achieve one of the key goals in grid computing – to allow users to get computing power as easily as one can get electrical power through a wall socket.

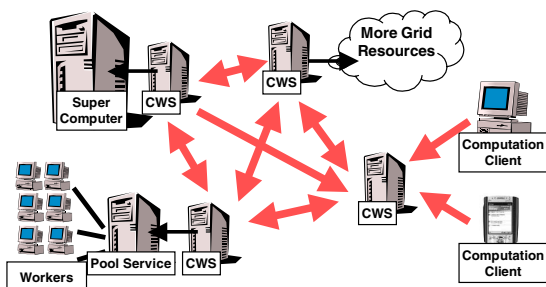


Figure 4. A grid of computational web services (CWS).

4 Results, Related Work, and Conclusion

Currently, we have used Bayesian Computing .NET to implement several simple volunteer computing applications, including a Mandelbrot set application, a ray-tracing application (based on Intel’s demo [3]), and a travelling salesperson (TSP) application. We have also implemented computational web services for the Mandelbrot and TSP applications, and have been able to use these with different kinds of computational clients including a .NET application running on a PC, a non-.NET Java applet, and a “grid portal” that can be accessed as a web page on any device with a web browser, including PDAs and cellular phones.

As far as we know, Bayesian Computing .NET, developed in September 2001, is the first generic system for volunteer computing and grid computing that uses web services. Intel has an earlier peer-to-peer cycle sharing demo [3], but it is not generic, and does not allow users to easily volunteer their machines. As far as we know, we are also the first to propose and to implement the idea of *computational web services*. Earlier work exists on *grid portals* that allow end-users to submit jobs to the Grid by filling up a form on a web page [4], but these are not as flexible and powerful as computational web services, which can be called like methods in other programs, and can thus be used not only by end-users, but by grid service providers as well, as shown in Fig. 4.

Our results are just the beginning of much possible research in this area. There are many areas in which grid computing and web services can be used together. Very recently, the Globus group has also started studying ways to integrate web services and grid computing [5]. Meanwhile, we plan to continue developing Bayesian Computing .NET as well by exploring issues such as authentication, authorization, resource discovery, and resource trading, and we hope that our results encourage further research by other as well.

References

- [1] Microsoft Corporation. XML and .NET White Papers. <http://www.microsoft.com/serviceproviders/whitepapers/xml.asp>
- [2] L.F.G. Sarmenta, *Volunteer Computing*, Ph.D. thesis. Massachusetts Institute of Technology, March 2001. <http://www.cag.lcs.mit.edu/bayanihan/>
- [3] B. Wilkerson. Grid Computing Using .NET Web Services and UDDI. Sept. 2001. <http://www.intel.com/ids/dotnet>
- [4] M. Thomas, et al. GridPort Launches Computational Science Applications on the Web. NPACI. April 2000. <http://www.npaci.edu/envision/v16.2/gridport.html>
- [5] I. Foster, et al. The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. Draft, Feb. 2002. <http://www.globus.org/>