Cluster Computing for Financial Engineering

Shirish Chinchalkar[1], Thomas F. Coleman[1], and Peter Mansfield[1]

CTC-Manhattan
Cornell University
55 Broad Street, Third Floor
New York, NY 10004, USA.
{shirish, coleman, peterm}@tc.cornell.edu

Abstract. The pricing of a portfolio of financial instruments is a common and important computational problem in financial engineering. In addition to pricing, a portfolio or risk manager may be interested in determining an effective hedging strategy, computing the value at risk, or valuing the portfolio under several different scenarios. Because of the size of many practical portfolios and the complexity of modern financial instruments the computing time to solve these problems can be several hours. We demonstrate a powerful and practical method for solving these problems on clusters using web services.

1 Introduction

The problems of financial engineering, and more generally computational finance, represent an important class of computationally intensive problems arising in industry. Many of the problems are portfolio problems. Examples include: determine the fair value of a portfolio (of financial instruments), compute an effective hedging strategy, calculate the value-at-risk, and determine an optimal rebalance of the portfolio. Because of the size of many practical portfolios, and the complexity of modern financial instruments, the computing time to solve these problems can be several hours.

Financial engineering becomes even more challenging as future 'scenarios' are considered. For example, hedge fund managers must peer into the future. How will the value of my portfolio of convertibles change going forward if interest rates climb but the underlying declines, and volatility increases? If the risk of default of a corporate bond issuer rises sharply over the next few years, how will my portfolio valuation be impacted? Can I visualize some of these dependencies and relationships evolving over the next few years? Within a range of parameter fluctuations, what is the worst case scenario?

Clearly such "what if" questions can help a fund manager decide today on portfolio adjustments and hedging possibilities. However, peering into the future can be very expensive. Even "modest" futuristic questions can result in many hours of computing time on powerful workstations. The obvious alternative to waiting hours (possibly only to discover that a parameter has been mis-specified), is to move the entire portfolio system to a costly supercomputer. This is a cumbersome, inefficient, and "user unfriendly" approach. However, there is good

news: most of these practical problems represent loosely-coupled computations and can be well solved on a cluster of processors in a master-worker framework.

We have been investigating the design of effective parallel approaches to the problems of financial engineering, and computational finance, on clusters of servers using web services. Our particular approach is to represent the portfolio in Excel with the back-end computing needs satisfied by a cluster of industry standard processors running in web services mode. The user environment we have used is Microsoft's .NET.

2 Introduction to web services

A web service is a piece of functionality, such as a method or a function call, exposed through a web interface ([1]). Any client on the internet can use this functionality by sending a text message encoded in XML to a server, which hosts this functionality. The server sends the response back to the client through another XML message. For example, a web service could compute the price of an option given the strike, the stock price, volatility, and interest rate. Any application over the internet could invoke this web service whenever it needs the price of such an option. There are several advantages in using web services to perform computations:

1. XML and HTTP are industry standards. So, we can write a web service in Java on Linux and invoke it from a Windows application written in C# and vice a versa.
2. Using Microsoft's .NET technology, we can invoke web services from office applications such as Microsoft Excel. This feature is especially useful in the financial industry, since a lot of end-user data is stored in Excel spreadsheets.
3. No special-purpose hardware is required for running web services. Even different types of computers in different locations can be used together as a web services cluster.
4. Since the web service resides only on the web server(s), the client software does not need to be updated every time the web service is modified. (However, if the interface changes, the client will need to be updated).
5. The web service code never leaves the server, so proprietary code can be protected.
6. Web services can be accessed from anywhere. No special purpose interface is necessary. Even a hand-held device over a wireless network and the internet can access web services.

In the context of large-scale financial computations, there are some challenges to web services as well:

1. There is no built-in mechanism to communicate with other web services. This limits the use to loosely coupled applications.
2. The web service can communicate with the client only at the end of the computation. For example, there is no mechanism for sending partial results while the computation is going on, without using another technology such as MPI.

3. Since messages are sent using a text format over the internet, this is not a viable computational technique for "short" computations involving a lot of data.

2.1 A simple web service

The following code shows a web service which adds two integers. It is written using the C# programming language. This code is written in the form of a class. When compiled it produces a dll which can be installed on the web server.

```
using System;
using System.Web.Services;
using System.Web.Services.Protocols;

namespace testnamespace
{
    public class testcls : System.Web.Services.WebService
    {
        [WebMethod]
        public int add(int a, int b)
        {
            return a + b;
        }
    }
}
```

This example shows that writing a web service is no different from writing a function or a method that performs the same computation. Other than a couple of declarative statements, there is no difference between a web service and an ordinary function. Notice that there is no reference to message passing, converting data into XML, and so on. These details are hidden from the programmer.

2.2 A simple main program

The following code shows a main program which accesses this web service and adds two numbers.

```
using System;
using testclient.localhost;

namespace testclient
{
    class Class1
    {
        [STAThread]
        static void Main(string[] args)
```

```
        {
            testcls t = new testcls();
            int a, b, c;
            a = 1;
            b = 2;
            c = t.add(a, b);
            Console.WriteLine("{0} + {1} = {2}", a, b, c);
        }
    }
}
```

Again, from this main program, it is evident that invoking the web service is no different from making a function call within the same process. Only one additional statement refers to the web service at the top of the program.

2.3 Cluster computing using web services

A typical portfolio manager could have a large portfolio of complex instruments. These instruments may have to be priced every day. Often, several scenarios of the stock market or interest rates may have to be simulated and the instruments may have to be priced in each scenario. Clearly, a lot of computing power is necessary. If the instruments can be priced independent of one another, we can make use of web services to perform this computation.

The entire computation can be partitioned into several tasks. Each task can consist of the pricing of a single instrument. We can have a separate web service to price each instrument. The client then simply needs to invoke the appropriate web service for each instrument. We can use other models of computation as well. For instance, in case of Monte Carlo simulation, we could split the simulations among the processors.

Figure 1 shows the overall organization of our architecture. The front-end is a typical laptop or a desktop running Excel. Data related to the portfolio is available in an Excel spreadsheet. This front-end is connected over internet or a LAN to a cluster of nodes, each of which runs a web server. When a large computation is to be performed, it is broken into smaller tasks by the Excel front-end. Each task is then shipped to an individual node which works on it independent of the other nodes. The nodes send results back to Excel, which is used to view results.

3 Load balancing

Given a set of tasks, we can distribute them across a .NET web services cluster in two different ways. We could send all the tasks at once to the main cluster node which uses Network Load Balancing (NLB) to distribute the tasks. However, the NLB monitors network traffic to determine which nodes are free and which are busy. This is suitable in transaction processing applications where each task

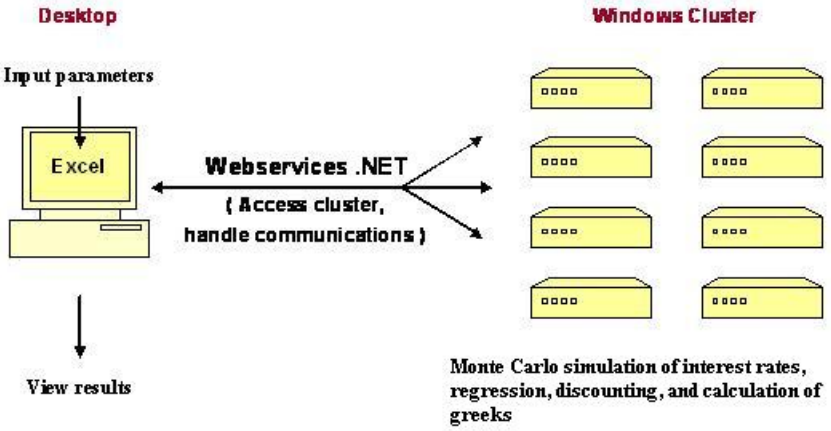

Fig. 1. Overview of computing architecture

can be processed fast, but the number of tasks is very large. For the problems we are interested in, we have a relatively small number of tasks, each of which takes seconds, minutes, or hours of computational time. For such problems the following approach is more suitable: We first assign tasks, one to each processor. When any task finishes, the next task is sent to the node which finished that task. This algorithm works well in practice provided there is only one application running on the cluster. If multiple applications need to run, a centralized manager is necessary.

The load balancing mechanism described above can be implemented as a class shown below, provided all data for all tasks is known before any task is executed. Fortunately, most finance applications that involve pricing portfolios of instruments fall in this category. By using a load balancing template, we can remove from the user application, most of the low-level "plumbing" related to multi-threaded operation. This makes applications significantly easier to program and promotes code reuse.

The following pseudo-code shows how such a load balancing class can be used inside a C# client application:

```
allargs = Array.CreateInstance(typeof(appinclass), numprobs);
for (int i=0; i<numprobs; i++)
{
    appinclass argtemp = new appinclass();

    // set arguments here
    // . . .
    // . . .

    allargs.SetValue(argtemp, i);
```

```
}
LBclass lb = new LBclass();
lb.allargs = allargs;
lb.serverURL = "http://hostname/webservice.asmx";
lb.numnodes = 4;
lb.run();

// wait for results
while (!lb.done) Thread.Sleep(50);
```

All code related to invoking the web service asynchronously on a multi-node cluster, determining free nodes, using locks for multi-threaded operation, sending inputs, receiving results, and generating timing and speedup information is handled by the class LBclass. If the user wishes to process results as and when they are returned, he needs to write an application specific callback, which is not shown above. Again, this callback does not involve any lower-level message passing related code.

4 An example

We show an example which involves the pricing of a portfolio of callable bonds. Although the type of instrument is not so important to demonstrate cluster computing using web services, we use this example to show that practical problems can be solved in this framework.

A typical corporate bond has a face value, a fixed coupon, and a maturity date. Such a bond pays a fixed amount of interest semi-annually until maturity. At maturity, the face value or principal is returned[2]. A callable bond has an additional feature - the bond may be 'called back' by the issuing company by offering the bond holder or the investor an amount equal to the face value of the bond. This buy-back can be made on any of the coupon payment dates. Whether the bond should be called or not depends on the prevailing interest rates and predictions of future interest rates. For example, if interest rates drop, it may be in the best interests of the issuing company to buy back the bond. If interest rates are high, the issuing company is unlikely to buy the bond. This presents two problems - first, future interest rates must be simulated, and second, the decision to buy the bond or not should be made at each coupon date, depending on the prevailing interest rate and the prediction of future interest rates.

For this work, we have used the Vasicek model for simulating interest rates. In this model, changes in interest rates are given by the formula

$$dr = a(\bar{r} - r)dt + \sigma dW \tag{1}$$

where dr is the change in the interest rate in a small time interval, dt, a is the mean reversion rate, $\bar{r}$ is the mean reversion level, and σ is the volatility. dW is a small increment of the Brownian motion, W (see [3] for more details). Given an initial interest rate, r_0, we can easily simulate future interest rates using the

above equation. For valuation of callable bonds and the calculation of greeks (see below), we need several tens of thousands of simulations.

Optimal exercise along each interest rate path is determined using the Least Squares Monte Carlo algorithm, which involves the solution of a linear regression problem at each coupon date and discounting of cashflows along each interest rate path. Details of this algorithm can be found in Longstaff and Schwartz[4].

We illustrate a few additional computations for a single bond. They can be extended to a portfolio quite easily. Along with the price of the bond, we also want the 'greeks' or bond delta and bond gamma. Delta is the first derivative of the bond price with respect to the initial interest rate ($\partial B/\partial r$) and gamma is the second derivative of the bond price with respect to the initial interest rate ($\partial^2 B/\partial r^2$), where B is the price of the bond. In this work, we have computed them using finite differences as follows

$$\Delta = \left.\frac{\partial B}{\partial r}\right|_{r=r_0} \approx \frac{B(r_0+dr) - B(r_0-dr)}{2dr} \tag{2}$$

$$\Gamma = \left.\frac{\partial^2 B}{\partial r^2}\right|_{r=r_0} \approx \frac{B(r_0+dr) - 2B(r_0) + B(r_0-dr)}{dr^2} \tag{3}$$

The above calculations require the pricing of the bond at two additional interest rates, $r_0 + dr$ and $r_0 - dr$. For all three pricing runs, we use the same set of random numbers to generate the interest rate paths (see [3]).

Once the greeks are computed, we can approximate the variation of the bond price by the following quadratic

$$B(r) \approx B(r_0) + \Delta(r - r_0) + \frac{1}{2}\Gamma(r - r_0)^2 \tag{4}$$

A risk manager would be interested in knowing how much loss this bond is likely to make, say, 1 month from now. This can be characterized by two metrics: Value at Risk (VaR) and Conditional Value at Risk (CVaR). These can be computed from the above approximation by another Monte Carlo simulation. For an introduction to VaR and CVar see [3].

The portfolio price, V, is simply a linear function of the individual bond prices

$$V = \sum_{1}^{n} w_i B_i \tag{5}$$

where the portfolio consists of n bonds, with w_i number of bonds of type i. The greeks can be computed analogously, and VaR and CVaR can be determined easily once the greeks are known.

Figure 2 shows the Excel front-end developed for this example. This interface can be used to view bond computing activity, cluster utilization and efficiency, a plot of portfolio price versus interest rate, and portfolio price, Value at Risk (VaR), Conditional Value at Risk (CVaR), and portfolio delta and gamma.

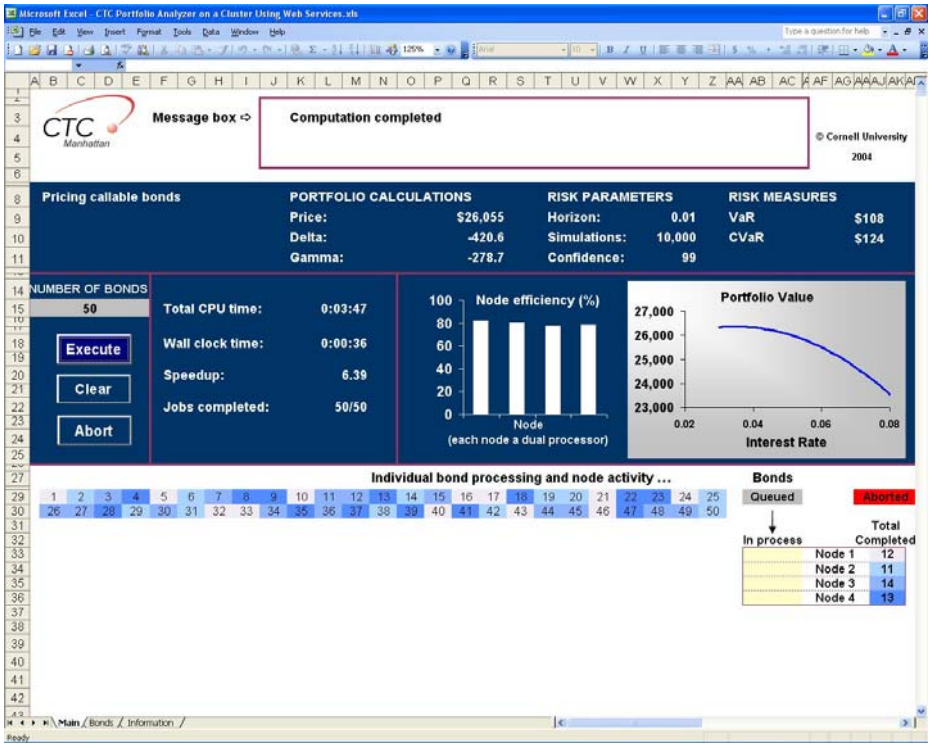

Fig. 2. Callable bond pricing in Excel

In our example, the web service computes the bond price and bond greeks, whereas the Excel front-end computes the portfolio price, greeks, VaR, and CVaR. Our experiments with portfolios of as few 50 instruments show that on 8 processors, we get speedups of 6.5 or more. On a 64 processor cluster, we have obtained speedups in excess of 60 for portfolios consisting of 2000 instruments.

5 Conclusion

Parallel computing, used to speed up a compute-intensive computation, has been under development, and in use by researchers and specialists for over a dozen years. Because a parallel computing environment is typically an isolated and impoverished one (not to mention very costly!) general industry has been slow to adopt parallel computing technology. Recent web services developments suggest that this situation is now improving, especially for certain application classes, such as portfolio modeling and quantitative analysis in finance. The work we have described here illustrates that a powerful analytic tool can be designed using web services technology to meet some of the computational challenges in computational finance and financial engineering.

6 Acknowledgements

This research was conducted using resources of the Cornell Theory Center, which is supported by Cornell University, New York State, and members of the Corporate Partnership Program. Thanks to Dave Lifka and his systems group at the Cornell Theory Center (Ithaca) for their support and to the other members of the CTC Computational Finance Group (Yuying Li and Cristina Patron) for their help.

References

1. A. Banerjee, et. al. *C# web services - building web services with .NET remoting and ASP.NET.* Wrox Press Ltd., Birmingham, UK, 2001.
2. W. Sharpe, G.J. Alexander, and J.W. Bailey. *Investments.* Prentice Hall. 1998.
3. P. Wilmott. *Paul Wilmott on Quantitative Finance, Volume 2.* John Wiley and Sons, New York, 2000.
4. F. Longstaff and E. Schwartz. Valuing American Options by Simulation: A Simple Least Squares Approach. *The Review of Financial Studies*, 14:113-147, 2001.