

Distributed Calculations on Fixed-Income Securities *

[Extended Abstract]

Timothy J. Williams
Argonne National Laboratory
9700 South Cass Avenue
Building 240
Argonne, IL 60439
zippy@anl.gov

ABSTRACT

This paper reviews real-world examples of distributed computing in the finance industry, specifically in institutional trading of fixed-income securities. Three examples illustrate small to large-scale distributed calculations: valuation of fixed-income derivatives, loading and producing time sequences of prices in statistical arbitrage on fixed-income instruments, and valuation of large portfolios of mortgage-backed securities. Two of these also serve to illustrate a recurring pattern in distributed access of financial data—distributed caching.

Categories and Subject Descriptors

J.1 [Administrative Data Processing]: Financial; D.1.3 [Programming Techniques]: Concurrent Programming—*Distributed Programming*; C.1.4 [Processor Architectures]: Parallel Architectures—*Distributed architectures*

Keywords

Fixed-income securities, pricing, valuation, mortgage-backed securities, distributed calculation, database, data cache

1. INTRODUCTION

These examples are taken from nine years' experience in technology in the finance industry—six years at Morgan Stanley in New York, and three years at Citadel Investment Group in Chicago. Prior to joining Morgan Stanley in 2000, I had spent eleven years in research as a computational scientist at the Department Energy's Lawrence Livermore and

*Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. WHPCF'09, November 15, 2009 Portland, Oregon, USA Copyright © 2009 ACM 978-1-60558-716-5/09/11... \$10.00

Los Alamos National Laboratories, specializing in massively parallel simulation of plasmas and fluids. That kind of fine-grained parallel computation was absent in all the work I had exposure to in finance, but there were numerous examples of coarser-grained distributed computation to be found. All of the work in this paper was the product of software development teams at the two financial institutions, whose members are to numerous to list (see acknowledgements in §4).

2. EXAMPLES

2.1 Fixed-Income Derivatives

In 2000, the fixed-income derivatives group at Morgan Stanley included both interest-rate and credit derivatives (primarily single-name Credit Default Swaps (CDS) in New York). Unix workstations were still pervasive on the trading desk, and Unix servers handled the heavy calculations. The trading system was designed as a monolithic, single-threaded server which maintained a large amount of state—all the state necessary for valuing interest-rate swaps and other derivatives. Trader screens ran a specialized client that communicated with the central server. While this was "distributed," the desk recognized that the monolithic server would not hold up indefinitely under increased trading. See Figure 1.

A development effort was already under way to augment this environment with multiple, concurrent calculation servers when I joined in April 2000. The clients would put XML messages bearing calculation requests and input parameters to a central message queuing system. A lightweight server would pick these up and distribute the calculations among a set of independent calculation servers. Results would be put back on a message queue, where they would be picked up by either the lightweight server (if they needed aggregation) or directly by the client. See Figures 2 and 3. All of these processes ran on Solaris[™] servers, displaying to Solaris desktops, though the change to displaying on X Windows running on Windows[®] desktop PCs was already underway.

Starting in late 2000, our group actively pursued porting the software to Linux, and by late 2001 our distributed calculation server system was one of the first two production applications running on Linux at Morgan Stanley. From that point on, the trend was strongly toward Linux for all servers.

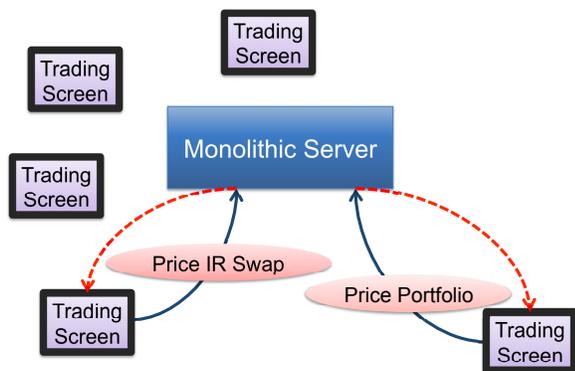


Figure 1: Schematic of original swaps trading components.

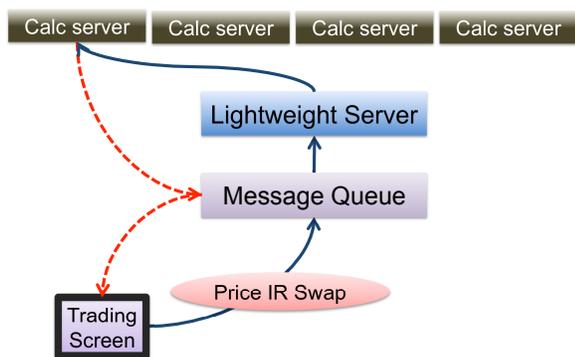


Figure 2: Distributed calculation of a single interest-rate swap price.

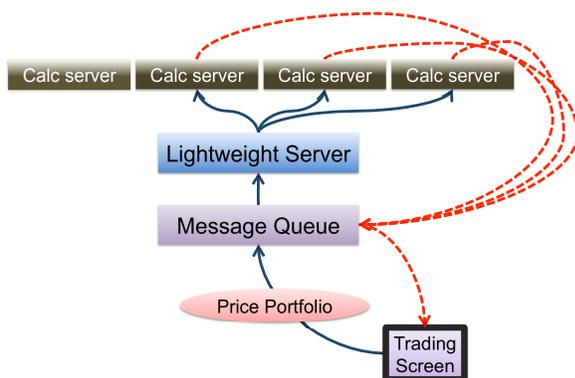


Figure 3: Distributed pricing of a whole portfolio of instruments. The server distributes the instruments among the available calculation servers.

For a year or so, we also had access to SGI[®] symmetric multiprocessor systems. There was some interest in "big iron" at the time, but our software mainly treated these machines as clusters of individual processors with no memory sharing. After that time, I wasn't aware of any subsequent interest in that architecture, or related tightly-coupled concurrent software development.

By 2005, Windows[®] desktops and clusters/racks of commodity Intel[®]-based Linux servers were the norm at Morgan Stanley. The interest-rate desk continued to use the same distributed calculation system. There were nightly risk calculations, as well as *ad hoc* security and portfolio valuations throughout the day. For intraday use, there were $O(10)$ calculation server nodes. One example of the various batch calculations: Each afternoon the desk would mark all the static yield curves (one or more per currency traded), and would switch on a cluster of $O(40)$ Linux blades to calibrate those curves.

During the years 2000 to 2005, microprocessor and compiler technologies had continued advance. By late 2005, a single processor running a rewritten curve calibration library was able to match the total wallclock compute time of the rack process described above. Basically, Moore's Law had shrunk the calculation time so that the overhead of processing XML messages in the distributed system became dominant. The messaging system, while based on industry practices when developed, was no longer appropriate for less heavyweight calculations like static yield curve calibration. Valuation of exotics and other compute-intensive stochastic calculations still benefited from the distributed system.

2.2 Statistical Arbitrage on Fixed-Income Instruments

In early 2006, I joined a team at Citadel Investment Group investigating strategies in statistical arbitrage on fixed-income instruments. Instruments traded included interest-rate futures contracts, bond futures, commodities futures, foreign-exchange futures, and others. Our models involved analysis of time series of prices. The quantitative research team ran various regressions on historical price sequences (daily and intraday). Models for trading acted on incoming time sequences of price quotes, and generated their own sequences of derived quantities which were persisted and used in future calculations. We used a small number of compute servers to handle incoming prices and compute derived quantities. These communicated via stripped-down hypertext transfer protocol (HTTP), and stayed synchronized via shared access to persisted data in databases.

The distributed calculation system was not complex or large-scale. More interesting was the maintaining of the time-sequence databases. We used the Sybase IQ[®] database for most storage, which is well suited to very "tall" tables like long time series. To keep up with incoming market data and our own ever-increasing derived quantities, we wound up using a file-based caching system that sat between the real database and some of the server processes. Appending to these binary files was fast, as was reading time sequences from them to bootstrap moving averages and other computed quantities. See Figure 4.

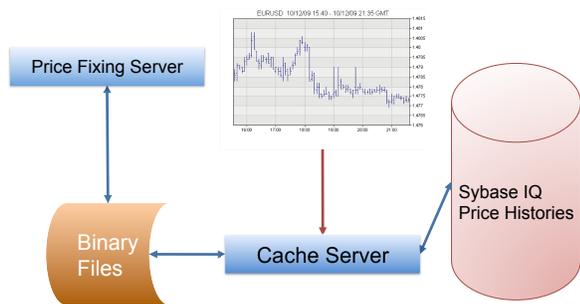


Figure 4: Managing time-series data. The price fixing server computed our own derived quantities and persisted their time series. The EURUSD graph represents various market price feeds. The cache server mediated between the file cache and the actual database.

2.3 Mortgage-Backed Securities

2.3.1 Calculations

In early 2007, I moved to the securitized products group at Citadel. There, we built and managed a large-scale distributed calculation system for valuing mortgage-backed securities (MBS), with emphasis on non-Agency MBS (including subprime and Alt-A). Refer to any of the standard literature for a discussion of the structure of an MBS and its valuation, for example Fabozzi[2, 3, 1].

Pricing these securities involves modeling the various market quantities that contribute to *prepayment* of the mortgages in the underlying pool. Various phenomena cause prepayment: relocation (selling the house prepays the whole mortgage and thus removes its future cashflows from the pool), delinquency (failing to make payments), and foreclosure, for example. One key factor is interest rates. If interest rates go down, there's an incentive for homeowners to refinance at a lower rate, which means their original mortgages are payed down and contribute no further cashflows to the MBS. If interest rates go up, owners of floating-rate mortgages may find their monthly payments too high to manage, leading to delinquencies and foreclosures. This was an especially strong factor in subprime mortgages at this time, because of the fixed/adjustable mortgages that reset to a very high spread over benchmark rates after some initial low fixed-interest rate for a few years.

Note that in these calculations we modeled the forward cash-flow behavior loanwise. That is, prepayment modeling was done at the loan level, based in part on some loan-specific factors. A typical deal has several thousand loans. Also note that all pricing was done via simulation, starting with an ensemble of $O(100)$ realizations of forward interest rates generated by a stochastic interest-rate model.

From a purely computational viewpoint, pricing a single MBS went as follows: For an ensemble of stochastic interest-rate paths, compute the forward cashflows of all the loans in the pool, using models to determine delinquencies and other prepayments that (in some paths) lead to payouts/losses from some mortgages. Feed these into the cashflow waterfall for the MBS to determine cashflows for the tranches.

Discount these cashflows to get their present value (PV). Average over all paths to get the overall PV for each tranche and, optionally, PV for all the loans. Compute interest-rate sensitivities (PV01) and other risk metrics based on these. Compute each *deal* (MBS) independently, which immediately produced significant embarrassing parallelism because the number of deals of interest was $O(1000)$. That parallelism formed the first basis for distributing the calculations.

Distribution was simple: For each deal, dynamically instantiate a pricer program on one processor, which loads all the data it needs, computes all the prices and risk metrics requested, and stages the results to files for uploading to a database. During the trading day, the desk would use a web interface to request *ad hoc* pricing of individual deals and specific portfolios.

The platform for distribution was commodity racks of commodity processors running Linux. The blades in these racks generally were connected with gigabit ethernet, but the connection fabric was not of much interest to us because of the embarrassingly parallel nature of the calculations. We had access to a *farm* of between 120 and 240 quad-core CPUs. In-house scheduling software dispatched calculation requests to physical processors. Web-based process monitors complemented the price-request interfaces. The trading desk used a desktop-based GUI for viewing the results.

Nightly risk calculations across all deals had even more inherent parallelism. Each deal would be priced for a set of $O(10)$ home price appreciation (HPA) scenarios, which could all be computed independently. Running on $O(700)$ cores, all the calculations for the $O(1000)$ deals of interest took about 4 hours of wallclock time.

Weekend runs added another factor of more than 10 in parallelism: running all the nightly calculations on $O(10)$ interest-rate-shift scenarios, and running some calculations on a larger universe of deals. Each weekend, running this batch of calculations on a farm of $O(700)$ cores took about 48 hours of wallclock time. Using a larger compute farm could clearly have sped this up if desired (but was not done, based on cost/benefit analysis).

We did not take advantage of two significant additional dimensions of parallelism in the problem: $O(100)$ independent simulation paths, and $O(1000)$ loans in each deal. Both of these are better suited for fine-grained parallelism using threading or multiprocessing with explicit control of inter-process communication and efficient communication volleys for aggregation over paths and loans.

2.3.2 Data

Pricing MBS is highly data-intensive, partly because of loading data and model parameters for each and every loan. The total number of data values for each loan is $O(100)$. This includes basic dates, characteristics, and balances for the loan, such as

- identifiers
- payment frequency

- origination date
- original amount
- maturity date
- initial rate
- loan-to-value ratio
- current rate
- index for floating rate
- ...

Also relevant for modeling prepayments is current loan status information such as

- delinquency status (30 days, 60 days, 90 days)
- loss amount

More refined models take into account qualitative information such as

- documentation on mortgage application
- property type

One of the most widely-used sources of loan-level data is the Loan Performance (LP) company. Our pricing used this data when available. In other cases, data generally takes the form of mortgage-originator-specific "tape" data. "Cracking tapes" to extract data from proprietary formats is common practice in the MBS industry. Our distributed calculation system, in order to take advantage of repeated requests for the same data, used a multi-level cache for loan data

The primary source was a central database into which LP and tape data were unified in our own schema. To augment this in the distributed environment, we deployed a small number of independent databases holding copies of loan data of current interest. We used MySQL for these, for convenience, and deployed them on some segregated Linux farm nodes. This layer addressed the problem of a flood of queries overwhelming the central database when performing large batch calculations.

To further speed up repeated access to the same data (as in repeated portfolio repricings during the day, or risk runs at night), we added in-memory cache processes. The scheduling server assigned deals round-robin style to these in-memory data caches. The cache process did the usual thing: if requested data was not in the memory cache, it fetched it from one of the distributed databases, then cached it for future reference. See Figure 5.

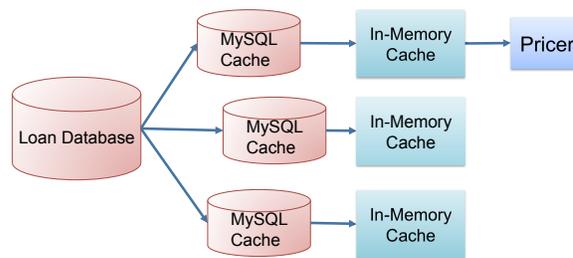


Figure 5: Loan data caching system.

3. OBSERVATIONS

Handling fast concurrent access to centralized loan data in §2.3.2 was very much like the problem addressed in §2.2. I know of similar homegrown solutions from a couple of other areas I had experience with: viewing pricing and risk results in CDS trading, and analyzing data from *tracking*, or historical-verification runs of prepayment models in MBS. Software offering some an appropriate general solution to this problem could have been put to good use in these cases, and probably elsewhere in this industry.

In MBS valuation, when the market comes back to life again, there may be a need to go beyond coarse-grained distributed calculations. Parallelizing across interest-rate paths, and across loans in a deal are the next logical steps. These would map well onto established techniques for fine-grained parallelism from the high-performance computing field.

4. ACKNOWLEDGEMENTS

The systems described in this paper are the fruit of the collective work of many people from Morgan Stanley and Citadel Investment Group. Where the paper alludes with general language to what are obviously specialized and significant software systems, it most likely alludes to the work of others who would be the appropriate people to publish specifics (in other cases, it indicates proprietary information not to be published). I would like to particularly acknowledge the interest-rate derivatives, credit derivatives, and FidMath teams at Morgan Stanley; the Mortgages group at Citadel; and also Richard Brooks and Slawomir Lisznianski, key architects of some of the software.

Windows is a registered trademark of Microsoft Corporation in the United States and other countries. SGI is a trademark [or service mark or registered trademark, as indicated] of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries. Solaris is a trademark or registered trademark of Sun Microsystems, Inc. in the United States and other countries.

5. REFERENCES

- [1] F. J. Fabozzi. *The Handbook of Mortgage Backed Securities*. Wiley, New York, New York, 2001.
- [2] F. J. Fabozzi. *Fixed Income Analysis, 2nd Edition*. Wiley, New York, New York, 2007.
- [3] F. J. Fabozzi. *Mortgage-Backed Securities: Products, Structuring, and Analytical Techniques*. Wiley, New York, New York, 2007.