City@home: Monte Carlo derivative pricing distributed on networked computers

Guido Germano*
Computer Simulation Group, Department of Chemistry
Philipps University Marburg, 35032 Marburg, Germany
E-mail: germano@staff.uni-marburg.de

Michael Engel
Distributed Systems Group, Department of Mathematics and Computer Science
Philipps University Marburg, 35032 Marburg, Germany
E-mail: engel@informatik.uni-marburg.de

Enrico Scalas
Department of Advanced Sciences and Technologies
Amedeo Avogadro University of East Piedmont, Via Bellini 25 G, 15100 Alessandria, Italy
E-mail: scalas@unipmn.it

Monte Carlo is a powerful and versatile derivative pricing tool, with the main drawback of requiring a large amount of computing time to generate enough realisations of the stochastic process. However, since realisations are independent from each other, the task is "embarassingly" parallel and the workload can be easily distributed on a large set of processors without the need for fast networking and thus an expensive dedicated supercomputer. Such an alternative, much cheaper and more accessible way can be realised with the BOINC toolkit, distributing the Monte Carlo runs on networked clients running under Windows, Linux or various Unix variants, and recollecting the results at the end for a statistical evaluation of the price distribution at the final time. Though it is likely that the clients will belong to the intranet of a large company or institution, we gave our program the evocative name City@home in honour of the paradigmatic SETI@home project. As an application, we present the generation of synthetic high frequency financial time series for speculative option valuation in the context of uncoupled continuous-time random walks (fractional diffusion), with a Lévy marginal density function for the tick-by-tick log returns and a Mittag-Leffler marginal density function for the waiting times. Lévy deviates are generated with the Chambers-Mallows-Stuck method, Mittag-Leffler deviates with the Kozubowski-Pakes method.

Grid Technology for Financial Modeling and Simulation
Palermo, Italy
3–4 February 2006

*Speaker.

PoS(GRID2006)011

1. Introduction

BOINC is an acronym for Berkeley Open Infrastructure for Network Computing [1]. It is a software platform for distributed computing that uses volunteered computer resources freely available under an open source license for various operating systems including Microsoft Windows, MacOS X, other Unix variants and Linux.

A typical PC spends most of its time doing nothing as it waits for user input or responses from peripheral devices or the network. These computing cycles spent idle can also be used to execute scientific calculations. While the spare computing capacity available from a single PC may be small, the combined power from millions of PCs connected to the internet allows to form a virtual supercomputer that can challenge the world's fastest systems.

Problems best solved on such a configuration are what is usually described as "embarassingly parallel": one large task that can be divided into a significant number of smaller, mostly independent calculations that can be completed by using the spare cycles of a single PC in a reasonable amount of time, i.e. from a few minutes to a few days.

The inspiration for BOINC came from the University of California at Berkeley's SETI@home project, that was one of the first projects featuring large-scale distributed computing on volunteers' PCs connected via the Internet. BOINC can be seen as a generalised platform based on the experiences of the SETI@home project, whose pioneering role motivated our name choice City@home for the particular application in mathematical finance that will be presented in the following.

2. Mathematical framework

2.1 Continuous-time random walks

Financial time series can be modelled phenomenologically as continuous-time random walks (CTRW) [2, 3, 4], also called point or renewal processes with reward. We consider the series of logarithmic returns $x(t) = \log(S(t)/S(0))$ of an asset price $S(t)$ with respect to its initial price $S(0)$. A CTRW is a jump process subordinated to a renewal process. In a renewal process, two consecutive events are separated by a random waiting time; events are a sequence of independent and identically distributed (i.i.d.) positive random variables τ_i:

$$t_n = t_0 + \sum_{i=1}^{n} \tau_i, \quad \tau_n = t_n - t_{n-1}, \quad n \in \mathbb{N}, \quad t_0 = 0. \tag{2.1}$$

The jump or reward process is a sequence of i.i.d. random variables ξ_i. Thus the position x of the random walker at time $t_n \leq t < t_{n+1}$ is:

$$x(t) = x(0) + \sum_{i=1}^{n} \xi_i. \tag{2.2}$$

CTRWs are good and general phenomenological models for diffusion, including anomalous diffusion, provided that the time of residence of the walker is much greater than the time it takes to make a jump; actually, in this formalism jumps are instantaneous.

In general, the jumps and the waiting times depend from each other, i.e. they have a joint probability density $\varphi(\xi,\tau)$, and the probability density $p(x,t)$ for the walker being in position x at time t, conditioned by the fact that it was in position $x=0$ at time $t=0$, is

$$p(x,t) = \delta(x)\,\Psi(t) + \int_{-\infty}^{+\infty}\int_0^t \varphi(x-x',t-t')\,p(x',t')\,dx'\,dt'. \tag{2.3}$$

The so-called survival function $\Psi(\tau)$ is related to the marginal waiting-time probability density $\psi(\tau)$. The two marginal densities are

$$\psi(\tau) = \int_{-\infty}^{+\infty} \varphi(\xi,\tau)\,d\xi, \qquad \lambda(\xi) = \int_0^{\infty} \varphi(\xi,\tau)\,d\tau, \tag{2.4}$$

and the survival function $\Psi(\tau)$ is

$$\Psi(\tau) = 1 - \int_0^{\tau} \psi(\tau')\,d\tau' = \int_{\tau}^{\infty} \psi(\tau')\,d\tau'. \tag{2.5}$$

The integral equation (2.3) can be solved in the Fourier-Laplace domain ($\widehat{f}(k) = \mathscr{F}_x[f(x)](k) = \int_{-\infty}^{+\infty} f(x)e^{ikx}\,dx$, $\widetilde{f}(s) = \mathscr{L}_t[f(t)](s) = \int_0^{\infty} f(s)e^{-st}\,dt$):

$$\widetilde{\widehat{p}}(k,s) = \frac{1-\widetilde{\psi}(s)}{s}\,\frac{1}{1-\widetilde{\widehat{\varphi}}(k,s)}. \tag{2.6}$$

In order to obtain $p(x,t)$, it is then necessary to invert its Fourier-Laplace transform $\widetilde{\widehat{p}}(k,s)$. A series solution exists for uncoupled CTRWs, where jump sizes do not depend on waiting times:

$$\varphi(\xi,\tau) = \lambda(\xi)\,\psi(\tau), \tag{2.7}$$

with the normalisation conditions $\int \lambda(\xi)d\xi = 1$ and $\int \psi(\tau)d\tau = 1$. In this case the master equation (2.3) for $p(x,t)$ becomes

$$p(x,t) = \delta(x)\,\Psi(t) + \int_0^t \psi(t-t')\,\left[\int_{-\infty}^{+\infty} \lambda(x-x')\,p(x',t')\,dx'\right]\,dt'. \tag{2.8}$$

This equation has a general explicit solution in terms of $P(n,t)$, the probability of n jumps occurring up to time t, and of the n-fold convolution $\lambda_n(x)$ of the jump density $\lambda(\xi)$:

$$\lambda_n(x) = \int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty}\dots\int_{-\infty}^{+\infty} d\xi_{n-1}d\xi_{n-2}\dots d\xi_1\,\lambda(x-\xi_{n-1})\lambda(\xi_{n-1}-\xi_{n-2})\dots\lambda(\xi_1). \tag{2.9}$$

Indeed, $P(n,t)$ is given by

$$P(n,t) = \int_0^t \psi_n(t-\tau)\Psi(\tau)\,d\tau \tag{2.10}$$

where $\psi_n(\tau)$ is the n-fold convolution of the waiting-time density:

$$\psi_n(\tau) = \int_0^{\tau}\int_0^{\tau_{n-1}}\dots\int_0^{\tau_1} d\tau_{n-1}d\tau_{n-2}\dots d\tau_1\,\psi(t-\tau_{n-1})\,\psi(\tau_{n-1}-\tau_{n-2})\dots\psi(\tau_1). \tag{2.11}$$

Its Laplace transform $\widetilde{P}(n,s)$ reads:

$$\widetilde{P}(n,s) = [\widetilde{\psi}(s)]^n\widetilde{\Psi}(s). \tag{2.12}$$

By taking the Fourier-Laplace transform of Eq. (2.8), one gets:

$$\widetilde{\widehat{p}}(k,s) = \widetilde{\Psi}(s)\,\frac{1}{1-\widetilde{\psi}(s)\widehat{\lambda}(k)}. \tag{2.13}$$

Since $|\widehat{\lambda}(k)| < 1$ and $|\widetilde{\psi}(s)| < 1$, if $k \neq 0$ and $s \neq 0$, Eq. (2.13) becomes:

$$\widetilde{\widehat{p}}(k,s) = \widetilde{\Psi}(s)\,\sum_{n=0}^{\infty}[\widetilde{\psi}(s)\widehat{\lambda}(k)]^n. \tag{2.14}$$

This gives, inverting the Fourier and Laplace transforms and taking into account Eqs. (2.9) and (2.10):

$$p(x,t) = \sum_{n=0}^{\infty} P(n,t)\lambda_n(x). \tag{2.15}$$

Once the log-price density $p(x,t)$ of an underlying is known, the pay-off density $q(y,t)$ of an option can be computed from $p(x,T)$ at maturity $t = T$ and from the pay-off function $y = \Pi(x,T)$ through the change of variable

$$q(y,T) = p[\Pi^{-1}(y,T),T]\left|\frac{dx}{dy}\right|. \tag{2.16}$$

E.g. for a plain vanilla call European option with exercise price E, the pay-off function is

$$y = \Pi(x,T) = \max\{\exp(x(T)) - E, 0\}. \tag{2.17}$$

Knowledge of the function $p(x,T)$ with parameters estimated from historic time series [5] allows speculative option valuation [6] through the computation of the expected payoffs, Eq. (2.16).

2.2 Choice of waiting-time and jump marginal densities

For the marginal waiting-time density we chose $\psi(\tau) = -d\Psi(\tau)/d\tau$, where $\Psi(\tau)$ is the Mittag-Leffler survival function with order $\beta \in (0,1]$ and scaling parameter γ_t:

$$\Psi(\tau) = E_\beta(-(\tau/\gamma_t)^\beta) = \sum_{n=0}^{\infty}\frac{\left(-(\tau/\gamma_t)^\beta\right)^n}{\Gamma(\beta n+1)}. \tag{2.18}$$

For the marginal jump density we chose a Lévy function with index $\alpha \in (0,2]$ and scaling parameter γ_x:

$$\lambda(\xi) = L_\alpha(\xi) = \mathscr{F}_k^{-1}\left[e^{-(\gamma_x k)^\alpha}\right](\xi) = \frac{1}{\pi}\int_0^\infty e^{-(\gamma_x k)^\alpha}\cos(\xi k)\,dk. \tag{2.19}$$

Thus jumps and waiting times are i.i.d. variables, and solving Eq. (2.15) for the density one gets:

$$p(x,t) = \sum_{n=0}^{\infty}\frac{(t/\gamma_t)^{\beta n}}{n!}E_\beta^{(n)}(-(t/\gamma_t)^\beta)\lambda_n(x). \tag{2.20}$$

This density can be computed exactly in the special case of the normal compound poisson process (NCPP), where $\alpha = 2$ and $\beta = 1$. With $\beta = 1$ the marginal waiting-time density becomes an exponential with $\mu = 1/\gamma_t$:

$$\psi(\tau) = \mu e^{-\mu\tau}; \tag{2.21}$$

with $\alpha = 2$ the marginal jump density becomes a normal with $\sigma = \sqrt{2}\gamma_x$:

$$\lambda(\xi) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\xi^2/2\sigma^2}. \tag{2.22}$$

The probability $P(n,t)$ of n jumps occurring up to time t is a Poisson distribution $e^{-\mu t}(\mu t)^n/n!$ and the n-fold convolution $\lambda_n(x)$ of the jump density is another normal with standard deviation $\sqrt{n}\sigma$, leading to the density

$$p(x,t) = e^{-\mu t} \sum_{n=0}^{\infty} \frac{(\mu t)^n}{n!} \frac{1}{\sqrt{2\pi n}\sigma} e^{-x^2/2n\sigma^2}. \tag{2.23}$$

Given the above choice of the marginal density functions, in the diffusive limit, i.e. $h \to 0$ and $r \to 0$ with the scaling relation $h^\alpha \sim r^\beta$, the CTRW density $p_{h,r}(hx,rt)$ converges to $u(x,t)$, solution of the following fractional diffusion problem [7, 8] with diffusion coeffi cient $D = \gamma_x^\alpha/\gamma_t^\beta$:

$$\begin{aligned} \frac{\partial^\beta}{\partial t^\beta} u(x,t) &= D \frac{\partial^\alpha}{\partial |x|^\alpha} u(x,t) \\ u(x,0^+) &= \delta(x), \quad x \in (-\infty,+\infty), \quad t > 0; \end{aligned} \tag{2.24}$$

$d^\alpha f(x)/d|x|^\alpha = \mathscr{F}_k^{-1}[-|k|^\alpha \widehat{f}(k)](x)$ is the Riesz space-fractional derivative of order $\alpha \in (0,2)$,

$$\frac{d^\alpha}{d|x|^\alpha} f(x) = \Gamma(1+\alpha) \frac{\sin(\alpha\pi/2)}{\pi} \int_0^\infty \frac{f(x+\xi) - 2f(x) + f(x-\xi)}{\xi^{1+\alpha}} d\xi, \tag{2.25}$$

and $d^\beta f(t)/dt^\beta = \mathscr{L}_s^{-1}[s^\beta \widetilde{f}(s) - s^{\beta-1} f(0^+)](t)$ is the Caputo time-fractional derivative of order $\beta \in (0,1)$,

$$\frac{d^\beta}{dt^\beta} f(t) = \frac{1}{\Gamma(1-\beta)} \left[\frac{d}{dt} \int_0^t \frac{f(\tau)}{(t-\tau)^\beta} d\tau - t^{-\beta} f(0^+) \right]. \tag{2.26}$$

In the case of marginal densities with a fi nite fi rst moment of waiting times and a fi nite second moment of log-returns, the limiting density $u(x,t)$ is the solution of the standard diffusion equation with $\alpha = 2,\ \beta = 1$, and thus the limiting process is a Wiener process.

3. Computational aspects

3.1 Program structure and non-uniform pseudorandom number generators

As explained in the previous section, an exact solution for the log-price density $p(x,t)$ exists only in the case of the NCPP, while series solutions with coeffi cients evaluated by numerical fast Fourier transforms are in order otherwise. Here we pursue the alternative to simulate by Monte Carlo the process given by Eqs. (2.1–2.2) and approximate $p(x,T)$ with a histogram of the log-prices $x(T)$ at maturity T. While the mathematics seen so far is diffi cult, a Monte Carlo code structure is straightforward: see Fig. 1.

Of course, the dust is hidden under the carpet of the random number generators `random_t` and `random_x`. However, we will see that these functions are quite easy to code and can be used as black boxes. As long as the random increments of each asset are independent of those of all other assets, the two outer loops can be parallelised trivially over $N \leq$ `assets*runs` processors: it is a so-called "embarassingly parallel" problem as referred to in the Introduction.

```
while (cin >> x_0 >> alpha >> beta >> gamma_x >> gamma_t) {
    histogram.zero();
    for (run = 0; i < runs; run++) {
        t = 0, x = x_0;
        while (t += gamma_t*random_t(beta) < t_max)
            x += gamma_x*random_x(alpha);
        histogram.add(x);
    }
    histogram.out();
}
```

Figure 1: Core of a C++ Monte Carlo program for the generation of continuous-time random walks.

Let $u \in (0,1)$ be a uniform pseudorandom variate; for its generation, we used the function `ran1` [9]. Non-uniform distributions can be generated by transformation from one u or two independent u_1, u_2: the exponential distribution taking $\tau = -\gamma_t \log u$ [9], the normal distribution with the modified Box-Muller method [9], the Mittag-Leffler distribution with the Kozubowski-Pakes method [10, 11, 12, 13]

$$\tau = -\gamma_t \log u_1 \left(\frac{\sin(\pi\beta)}{\tan(\pi\beta u_2)} - \cos(\pi\beta) \right)^{\frac{1}{\beta}} \tag{3.1}$$

that is a generalisation of the transformation method for the exponential (it reduces to the latter for $\beta = 1$), and the Lévy distribution with the Chambers-Mallows-Stuck method [14, 15, 16]

$$\xi = \gamma_x \left(\frac{w \cos\phi}{\cos((1-\alpha)\phi)} \right)^{1-\frac{1}{\alpha}} \frac{\sin(\alpha\phi)}{\cos\phi}, \quad \phi = \pi\left(u_1 - \frac{1}{2}\right), \quad w = -\log u_2, \tag{3.2}$$

that is a generalisation of the Box-Muller method (it reduces to the latter for $\alpha = 2$).

Our aim is to simulate a whole financial market with several hundred traded assets for a duration T between one day and one month. We started our tests with one asset traded for $T = 10\,000$ seconds, i.e. about three hours, and a parameter $\gamma_t = 10$ seconds. This means that events happen approximately every 10 seconds, i.e. there are about 1000 events (Figs. 2–4). A reasonably smooth histogram requires about 100 000 Monte Carlo runs (Fig. 2 right) and therefore 10^8 random waiting times and 10^8 random jumps. The CPU time needed to generate 10^8 pseudorandom numbers on different architectures for different distributions with a C++ program of ours is reported in Table 1.

There may be faster methods for the generation of non-uniform pseudorandom numbers like the Ziggurat [17], but the setup of the latter for the Mittag-Leffler and Lévy distributions is difficult, and the speed gain is modest compared to the loss when using e.g. simple rejection, especially if the Mittag-Leffler density is computed by a series expansion according to its definition, Eq. (2.18), for every point that is tested: up to 200 terms of the series are needed to achieve an acceptable accuracy [18], and rejection done this way turns out more than 5000 times slower than the Kozubowski-Pakes method! Simple rejection with a series evaluation of the Lévy density is "only" 400 times slower than the Chambers-Mallows-Stuck method. The latter and its more recent and less well known

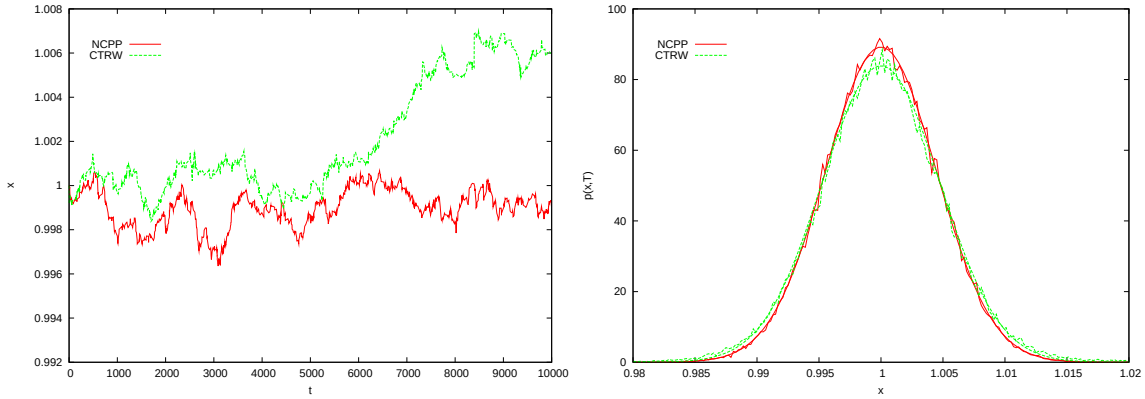

Figure 2: Left: time series for $\alpha = 2$, $\beta = 1$ (red) and $\alpha = 1.90$, $\beta = 0.97$ (green). In both cases $\gamma_x = 0.0001$, $\gamma_t = 10.0$, $T = 10\,000$. Right: histograms at $T = 10\,000$ and Gaussian fits.

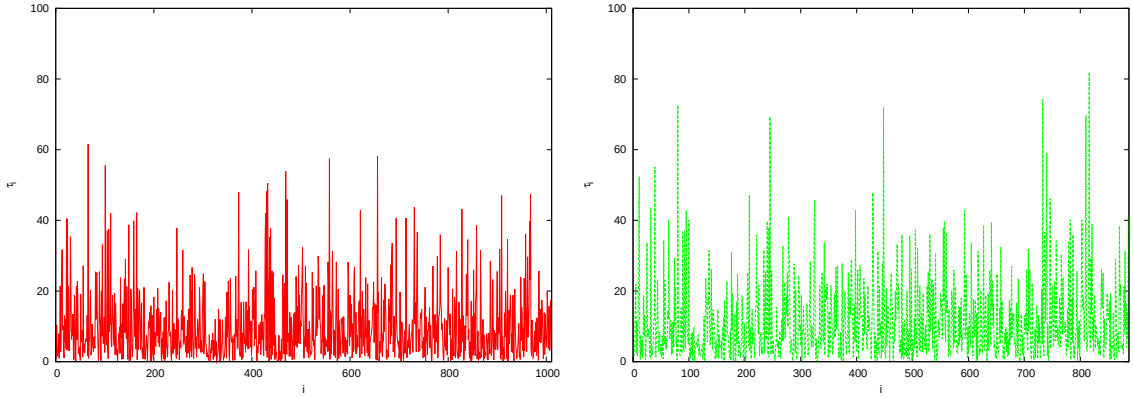

Figure 3: Waiting times for $\alpha = 2$, $\beta = 1$ (left) and $\alpha = 1.90$, $\beta = 0.97$ (right). Notice that the waiting times of the CTRW (right) are longer and therefore fewer.

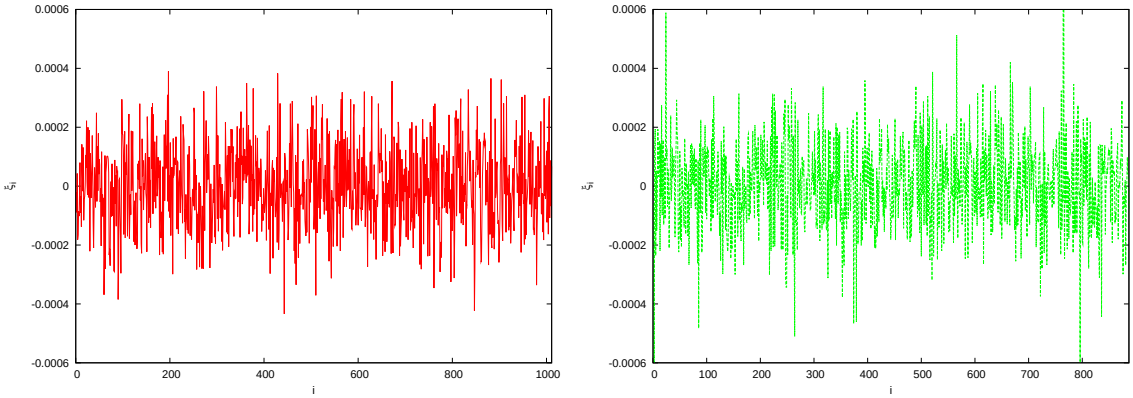

Figure 4: Jumps for $\alpha = 2$, $\beta = 1$ (left) and $\alpha = 1.90$, $\beta = 0.97$ (right). Notice that the jumps of the CTRW (right) are bigger (the vertical scale is truncated).

companion by Kozubowski and Pakes deal with remarkable simplicity and smartness with the awkward Lévy and Mittag-Leffler distributions, that do not have analytical expressions. We find these methods most adequate for our purposes. At least they avoid blunders like simple rejection connected with a series evaluation for every point. Even with an old Pentium IV processor, 100 000 high-frequency simulations of a whole trading day with a demanding model like a CTRW can be achieved in about 5 minutes, i.e. only 4 times more than with a standard NCPP.

	Pentium IV	Athlon 64	Opteron 270	IBM Power4+
Exponential	16	11	12	20
Gaussian	16	12	11	19
Mittag-Leffler	52	44	36	72
Lévy	73	66	52	95

Table 1: CPU time in seconds needed to generate 10^8 pseudorandom numbers on different architectures for different probability distributions. The Pentium IV operates at 2.4 GHz, the Athlon 64 (an X2 Dual-Core) at 2.2 GHz, the Opteron 270 at 2.0 GHz, and the Power4+ at 1.7 GHz. On the first three architectures we used the Intel compiler with the -O3 optimisation option; on the IBM, we used the xlC compiler with the -O5 option.

Though we have seen that a clever choice of the random number generators makes the model treatable on a modest desktop computer, parallelisation can still be used to get results in a few seconds rather than a few minutes: distributing the runs on $N \ll$ `runs` processors yields an almost linear speed-up, i.e. the wallclock time decreases by the same factor N. Parallelisation becomes more interesting for a whole market with hundreds of assets. We have experimented parallelising over both the number of runs and the number of assets, but eventually we concentrated on just the latter approach, because it is simpler and it is acceptable for us to wait minutes for the results. This might be different in a bank.

3.2 Virtual versus dedicated parallel computing

We consider two notable parallelisation paradigms. The first and most well-known for over a decade consists in resorting to the Message Passing Interface (MPI) [19] on a cluster of Linux PCs or Unix workstations or a dedicated parallel computer, e.g. our own Linux cluster with 85 dual core dual Opteron 270 nodes or the IBM Regatta p690+ and BlueGene/L supercomputers installed at Forschungszentrum Jülich in Germany [20]. Only C++ and Fortran 95 are supported on the latter platforms.

A more recent alternative consists in using the Berkeley Open Infrastructure for Network Computing (BOINC) [1] on heterogeneous hardware, most notably Windows PCs found in large organisations like universities (computer rooms for students) and companies (trading floors), or volunteered by peers over the Internet. The middleware runs like a screen saver. Our name City@home was mutuated from the oldest and most well-known application, SETI@home (search for extraterrestrial intelligence by radio evidence). Other BOINC applications are Folding@home (investigation of protein folding), Predictor@home (study of protein-related diseases), cellcomputing.org (biomedical research), worldcommunitygrid.org (advance knowledge of human diseases), Rosetta@home (develop cures for human diseases), Einstein@home (search for gravitational sig-

nals emitted by pulsars), LHC@home (improve the design of the CERN large hadron collider), climateprediction.net (study of climate change), Quantum Monte Carlo at home, Grid.org, distributed.net, etc.

A comparison between dedicated and virtual high performance computers shows interesting results. The IBM Regatta p690+ with its $32 \times 41 = 1312$ Power4+ processors running at 1.7 GHz has 8.9 Terafbps peak performance, 5.6 Terafbps sustained performance, 56 Terabytes disk space, and 1200 Terabytes tape archive; a few months after its installation in early 2004, it ranked 1st in Europe and 21st in the world [21]. Its cost was 42 million €, plus 5 million € for the building. Volunteer or peer-to-peer or public resource computing currently spreads half a dozen most popular applications over about 1 million hosts (0.1% of estimated existing PCs), providing a sustained performance of 95.5 Terafbps (60 Terafbps for SETI@home alone) and 7740 Terabytes of storage with an access rate of 5.27 Terabytes/second [22]. At zero cost, this surpasses even the new BlueGene/L with 45 Terafbps peak power that has been recently installed in Jülich as the most powerful supercomputer in the world for exclusively civil usage (it is the 6th on the absolute scale) [21] and is dedicated only to a small number of selected scientific grand challenges.

In the initial stage of our project, our testbed is restricted to a small number of local computers, and of course even in a later stage it will not be neither likely nor necessary to harness as much volunteered computer power as SETI@home. However, we consider the BOINC approach interesting per se. By the way, running on local trusted machines is more efficient by a factor 2 because redundancy becomes superfluous: many applications use this stratagem to minimise the effect of malfunctioning or malicious anonymous hosts. Each task is executed on two hosts belonging to different volunteers; if the results agree within a tolerance, they are accepted, otherwise a third instance is executed, and so on. If the fraction of inconsistent results is low, redundant computing reduces effective computing power by a factor of only slightly more than 2.

4. BOINC components

The large-scale distributed nature of BOINC applications requires elaborate coordination of the various system components. Since BOINC aims to be a generic framework for volunteer-based compute projects, most of the architecture can be reused for a large range of applications. As a consequence, a flexible, modular client-server architecture is provided into which the specific application can be integrated. In this section, we give a short overview of the various BOINC components on server as well as on client side (Fig. 4).

4.1 Server side

The most important component on the server side is the project back end. The back end is the central unit in charge of coordinating a BOINC project. It provides applications and work units to systems participating in a specific project. Furthermore, it handles all communication with the clients, i.e. it receives, checks and manages the results generated by all client systems.

As a separate component, mostly to reduce network load on the project back end side, data servers manage the distribution of work packages. The data servers must not be part of the project infrastructure itself, and can also be operated at different locations to avoid data throughput bottlenecks.

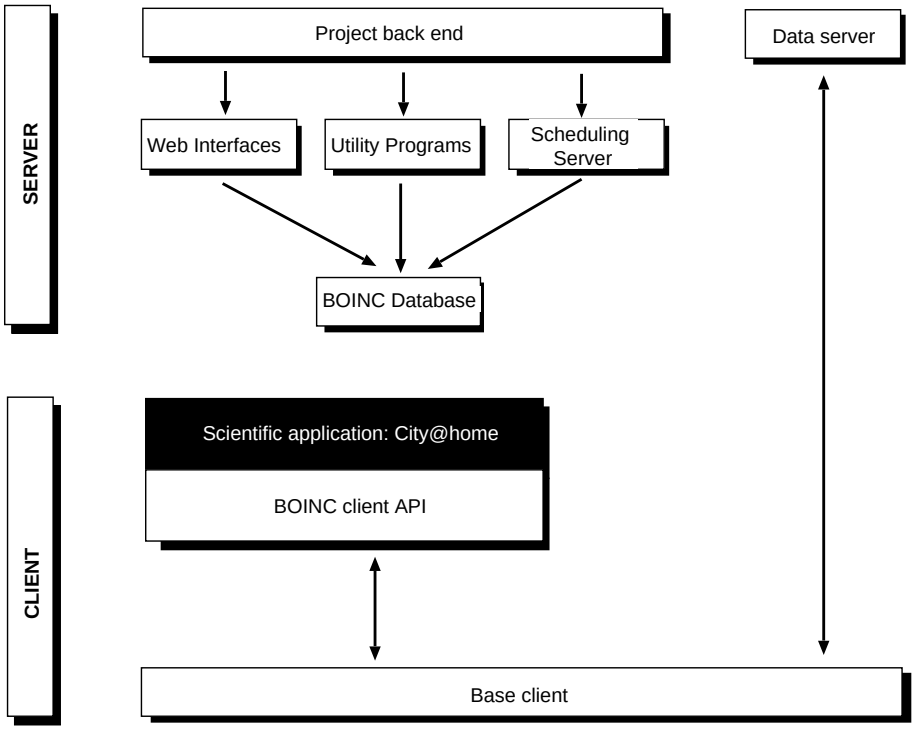

Figure 5: Structure of City@home distributed using BOINC.

The BOINC server complex consists of four components: the web interfaces, utility programs, scheduling servers, and the BOINC database. The web interfaces are the primary user interfaces for BOINC applications. They are usable from any web browser and are separated for project developers and participants. While participants can obtain information about a specific project's aim, software releases and current statistics, administrators are provided with progress reports and are enabled to take actions. The utility programs act as interface between the server complex and the project back end. One or more scheduling servers communicate with the clients. They are responsible for coordinating the work that is ready to issue so as to make the best use of the computers available. Finally, the BOINC database stores information about work packages, results and projects participants in a relational MySQL database.

4.2 Client side

On the client side of the BOINC infrastructure, a modular framework is provided that permits application developers to concentrate on their specific application by providing and encapsulating all communication requirements in client software components common to all BOINC-based projects.

The base client is the elementary BOINC component on the computer of each project participant. It communicates with the scheduling server, fetches new work units, delivers the results back and buffers a certain amount of work packages and results in a local cache. Depending on the number of processors available — that differ over time — the work units are kept comparatively small. In our case, with typical values for `assets` of a few 100 and for `runs` of about 100 000,

a workload unit consists of $1-10\times100\,000$ runs of the inner while-loop, i.e. of the complex part of the calculation that calls the `random_t` and `random_x` functions. Each client computer is assigned a unique ID when logging in to a BOINC server for the first time. All further connections to the server are enumerated using this ID and stored on client and server side. The base client also acquires and manages information on the local system like number and type of CPUs, available disk space, amount of main memory etc. This data is stored in the BOINC database and form the basis for deciding if a particular client will be handed a certain task or not. The statistical evaluations of Ref. [22] used this data.

One base client is able to support and manage several scientific applications at the same time, so no installation overhead is generated for a potential BOINC user who wishes to provide computing power to several projects simultaneously.

On the client side, there are several APIs available. The base client provided by the BOINC infrastructure and the scientific BOINC application communicate using the BOINC API as well as a Graphics API. The base client generates a graphical progress indicator for each project. In addition, the project application can deliver more complex graphical output and even complete screen savers. Often, OpenGL is used for graphics output; the base client and the applications communicate via a BOINC-specific XML-based protocol.

Finally, the scientific application is the specialised distributed computational problem to be solved. It consists of a program (typically compiled for several client platforms like Windows, MacOS X and Linux/x86), work units and results. A project may consist of several separate applications that take care of the scientific calculations.

Scientific applications must not necessarily be developed as open source components, even though BOINC itself is an open source project. This provides developers of large-scale distributed applications with the advantage of being able to keep algorithms and data relatively secure, though no real security mechanisms against reverse-engineering the client-side application are provided by BOINC. An approach to categorise the possible threats for applications in a large-scale distributed computing environment can be found in [23]. Many of the threat models discussed in this paper are also relevant for BOINC-based applications; however, in our current intranet-based deployment of City@home, security considerations are not an issue.

5. Conclusions

BOINC provides a flexible, no-cost framework for developing distributed computing applications. Due to its support of C/C++ and Fortran code as applications, existing code may be leveraged and reused. BOINC is mostly suited for applications involving easily parallelisable parts. For these kind of applications, thanks to the power of idle PCs either in one's own organisation or volunteered over the internet, BOINC-based calculations experience a significant speedup.

Even with such a computationally expensive model like a Monte Carlo CTRW, large portfolios can be simulated in a few minutes within intra-day scenarios of synthetic high-frequency time series, depending on how many processors are available.

6. Acknowledgments

We thank the John von Neumann Institute for Computing in Jülich for access to its IBM Regatta p690+ parallel computer JUMP and for support, A. Vivoli for providing us his scalar CTRW Monte Carlo simulation code in Fortran 90 with series evaluation of the Lévy and Mittag-Leffler densities, and B. Böttcher and D. Fulger for finding the Kozubowski-Pakes algorithm.

References

[1] The BOINC project, `http://boinc.berkeley.edu`.

[2] E. Scalas, R. Gorenflo, H. Luckock, F. Mainardi, M. Mantelli, M. Raberto, *Anomalous waiting times in high-frequency financial data*, *Quant. Finance* **4** (2004) 695.

[3] E. Scalas, R. Gorenflo, F. Mainardi, *Uncoupled continuous-time random walks: Solution and limiting behavior of the master equation*, *Phys. Rev. E* **69** (2004) 011107.

[4] E.Scalas, *The application of continuous-time random walks in finance and economics*, *Physica A* **362** (2006) 225.

[5] A. Tedeschi, *High frequency financial data in eGrid*, in proceedings of the *1st International Workshop on Grid Technology for Financial Modeling and Simulation*, Palermo, 3–4 February 2006, edited by S. Cozzini, S. d'Addona, R. Mantegna, PoS(GRID2006)016, `http://pos.sissa.it`.

[6] E. Scalas, R. Gorenflo, F. Mainardi, M. M. Meerschaert, *Speculative option valuation and the diffusion equation*, in proceedings of the *1st IFAC Workshop on fractional differentiation and its applications*, Bordeaux, 19–21 July 2004, edited by A. Le Mehauté, J. A. Tenreiro Machado, J. C. Trigeassou, J. Sabatier, p. 265, U Books 2005.

[7] A. I. Saichev, G. M. Zaslavsky, *Fractional kinetic equations: solutions and applications*, *Chaos* **7** (1997) 753.

[8] R. Gorenflo, A. Vivoli, F. Mainardi, *Discrete and Continuous Random Walk Models for Space-Time Fractional Diffusion*, *Nonlinear Dinam.* **38** (2004) 101.

[9] W. H. Press, S. A. Teukolsky, W. T. Vetterling, B. P. Flannery, *Numerical Recipes in C++ — The Art of Scientific Computing*, 2nd Ed., Cambridge University Press 2003.

[10] A. G. Pakes, *Mixture representations for symmetric generalized Linnik laws*, *Stat. Probabil. Lett.* **37** (1998) 213.

[11] T. J. Kozubowski, *Mixture representations of Linnik distribution revisited*, *Stat. Probabil. Lett.* **38** (1998) 157.

[12] T. J. Kozubowski, S. T. Rachev, *Univariate Geometric Stable Laws*, *J. Comput. Anal. Appl.* **1** (1999) 177.

[13] T. J. Kozubowski, *Fractional Moment Estimation of Linnik and Mittag-Leffler Parameters*, *Math. Comput. Model.* **34** (2001) 1023.

[14] J. M. Chambers, C. L. Mallows, B. W. Stuck, *A Method for Simulating Stable Random Variables*, *J. Am. Stat. Assoc.* **71** (1976) 340.

[15] J. H. McCulloch, *stabrnd.m* MATLAB function, Economics Department, Ohio State University, `http://economics.sbs.ohio-state.edu/jhm/jhm.html`.

[16] R. Weron, *On the Chambers-Mallows-Stuck method for simulating skewed stable random variables*, *Stat. Probabil. Lett.* **28** (1996) 165;
R. Weron, Correction, `www.im.pwr.wroc.pl/~hugo/publ/RWeron_HSC_96_1.pdf`.

[17] G. Marsaglia, W. W. Tsang, *The Ziggurat Method for Generating Random Variables*, *J. Stat. Software* **5** (2000) 8, `http://www.jstatsoft.org/v05/i08/ziggurat.pdf`.

[18] A. Vivoli, *Non-Gaussian Stochastic Processes and Their Applications*, Laurea thesis (in Italian), University of Bologna 2002.

[19] The Message Passing Interface, `www.mpi-forum.org, www-unix.mcs.anl.gov/mpi.`

[20] IBM p690+ e-server JUelich Multi Processor (JUMP), `http://jumpdoc.fz-juelich.org.`

[21] Top 500 supercomputer sites, `www.top500.org`.

[22] D. P. Anderson, G. Fedak, *The Computational and Storage Potential of Volunteer Computing*, in proceedings of the *6th IEEE International Symposium on Cluster Computing and the Grid 2006 (CCGrid06)*, Singapore, 16–19 May 2006, edited by S. J. Turner, B. S. Lee, W. Cai, Vol. 1, p. 73, IEEE Press 2006, `http://boinc.berkeley.edu/boinc_papers`.

[23] M. Smith, M. Engel, T. Friese, B. Freisleben, G. A. Koenig, W. Yurcik, *Security Issues in On-Demand Grid and Cluster Computing*, in proceedings of the *2nd International Workshop on Cluster Security (Cluster-Sec'06)* at the *6th IEEE/ACM International Symposium on Cluster Computing and the Grid 2006 (CCGrid06)*, Singapore, 17 May 2006, edited by S. J. Turner, B. S. Lee, W. Cai, Vol. 2, p. 24, IEEE Press 2006, `http://pdcc.ntu.edu.sg/ccgrid2006`.