EGG: AN EXTENSIBLE AND ECONOMICS-INSPIRED OPEN GRID COMPUTING PLATFORM

J.BRUNELLE[*], P.HURST[‡], J.HUTH[‡], L.KANG[†], C.NG[†], D.C.PARKES[†], M.SELTZER[†], J.SHANK[‡] and S.YOUSSEF[§]

[*]*Department of Physics, Boston University, Boston, USA*
[†]*DEAS, Harvard University, Cambridge, USA*
[‡]*Department of Physics, Harvard University, Cambridge USA*
[§]*Center for Computational Science, Boston University, Boston, USA*

The *Egg* project provides a vision and implementation of how heterogeneous computational requirements will be supported within a single grid and a compelling reason to explain why computational grids will thrive. Environment computing, which allows a user to specify properties that a compute environment must satisfy in order to support the user's computation, provides a how. Economic principles, allowing resource owners, users, and other stakeholders to make value and policy statements, provides a why. The Egg project introduces a language for defining software environments (*egg shell*), a general type for grid objects (the *cache*), and a currency (the *egg*). The Egg platform resembles an economically driven Internet-wide Unix system with egg shell playing the role of a scripting language and caches playing the role of a global file system, including an initial collection of devices.

1. Introduction

Although grid software and its conceptual framework have evolved substantially in recent years, today's grids are still limited in size to hundreds of sites and thousands of computers. They have not approached Internet-scale. Grids have not grown larger and are not used more broadly due to five factors: 1) There is no global resource allocation mechanism. 2) Installing and maintaining grid infrastructure software is time-intensive and difficult. 3) Converting applications to be grid-enabled is also time-intensive and difficult. 4) It is complex to express user and organizational policies. 5) It is difficult, often impossible for users to express needs.

Grid computing will become mainstream only when its existence can be largely ignored. For example: when researchers can focus solely on computational experiments and not on the tools, configurations, and gaming of the system required to obtain sufficient resources, grid computing will be

more popular. Similarly, when two organizations can share resources in a flexible and locally managed manner, those organizations are more likely to work together and share those resources.

We introduce *Egg*, a system that provides nearly transparent and natural access to grid resources. The Egg architecture brings together environment computing and economic principles to create a vision of grid computing that realizes the promise of autonomy and openness.

2. Egg Overview

Egg is an extensible, economics-based platform for open grid computing. Egg addresses the shortcomings of the existing grid solutions by focusing on the following design principles:

(1) **Simplicity and Extensibility**: Every grid function, for example, submitting jobs to run, storing files, assigning user permissions, should involve only a small set of standard operations, such as *put* and *get*.
(2) **Smart Resource Allocation**: Users can describe their jobs well in a language of computational experiments, but not in terms of characteristics of the machines that can run those jobs. Transparent resource discovery and allocation that are in the best interests of the users simplify interactions.
(3) **Decentralized Policy and Control**: All stakeholders must have a flexible and easily comprehensible way of influencing the overall system. Organizations with different resources can share resources by creating realistic economies while still retaining as much control as desired over policy.

All three principles are critical and are interrelated. Individually, each represents a significant breakthrough in the respective problem domain in grid computing. The three major components of Egg correspond to these principles: *egg shells* and *caches*, *microeconomic architecture*, and *macroeconomic architecture*. We discuss each in detail.

3. Fundamental Architecture

3.1. *Egg shells*

Egg shell is a language that allows a user to specify both the environment requirements, to be established via installation, together with the details of the computational job. This is the essential element of what we term

environment computing. The details provided by a user can include information such as the location of source data and the name and location of the analysis program that is to be executed.

In fact, the inspiration for egg shell comes from the humble activity of installing software.[1] Egg shell is a small superset of the Pacman language, which is fast becoming the *de facto* standard for deployment of software environments on grids in the US research community. The Open Science Grid (OSG)[2] is entirely deployed with Pacman[3], as is the Virtual Data Kit, including installation of Condor and Globus and other grid middleware[4], as is ATLAS[5] and other experiments.[6]

There is one key egg shell primitive. This primitive, `put`, is used to add one type of cache to another type of cache and can be thought of as a generalized copy. For instance, `put` initiates a job, downloads and uploads data, or installs a software environment. Other egg shell primitives, such as `pay` can be defined in terms of `put`. The `pay` primitive initiates payment for a service rendered, and is defined as a `put` of egg currency (the *egg*) to a bank cache. The following example egg shell demonstrates the simplicity and expressivity of egg shell.

```
1: { put ~Alice/jobs/runningEnvironment.eggshell mygrid OR fail
         "Can't run on this machine." }
2: put ~Alice/jobs/binary1 mygrid
3: FOR j in [job1.eggshell,job2.eggshell,job3.eggshell] {
4:          put ~Alice/jobs/j mygrid
5:          put results/j.out ~Alice/results
6:          pay @100.Harvard.Alice when gmTime < 1-Apr-2006
7: }
8: shell echo "done"
```

Currency is denoted with `@` as a prefix, e.g. `@20.Harvard.Alice` denotes 20 units of Harvard currency held by Alice. Notice the conciseness of egg shell. In just 8 lines, it captures the configuration of the environment (line 1), dispatching jobs to machines (line 4), and payment terms (line 6).

[1] Pacman web site: (http://physics.bu.edu/pacman/).
[2] `www.opensciencegrid.org`
[3] See: `http://kb.grid.iu.edu/data/aths.html`
[4] `http://vdt.cs.wisc.edu/`
[5] The ATLAS project is a massive collaborative effort in particle physics, involving over 1800 physicists from more than 150 universities and laboratories in 34 countries. `http://atlas.web.cern.ch/Atlas/index.html`
[6] `http://atlas.bu.edu/caches/registry/E/htmls/registry.html`

A great deal of complexity and autonomy underlies the simplicity of egg shell. Line 1 (`put Alice/jobs/runningEnvironment.eggshell`) refers to a typical installation script.[7] Line 4 (`put Alice/jobs/j mygrid`) runs a job, without requiring that the user specify machines on which to run it or any characteristics about the job. The characteristics of the job are determined by agents local to caches, that learn these characteristics by observing the characteristics of past jobs. The payment terms in line 6 (pay @100.Harvard.Alice) provide all the guidance needed for the system to determine the best machine for the job, and indeed predict whether the job can be completed by the deadline.

3.2. *Caches*

All functional elements of Egg are *caches* including computers of various kinds, storage devices, bidding agents, files, directories, egg shell source and object files, banks, marketplaces, rolodexes printers, garbage cans etc. A cache can be thought of as a box with an input port that supports an operation of *put*-ing one cache into another. Caches have internal existence as python objects in the Egg system and an external existence as bindings to URLs or servers. The relationship that a cache has to the external world is maintained by a pair of functions *eval*/*save* which are generalizations of *read*/*write* combining i/o, search operations, lazy evaluation and individual cache-specific computations. For example, if an egg shell is *put* into a computer cache, the *save* operation may attempt to execute the egg shell. The same egg shell put into a bidding cache might, on the other hand, cause a bid to be constructed; a banking cache might strip the egg shell of currency and a storage cache might simply store the egg shell as a file.

4. Microeconomic Architecture

In Egg, we provide *smart resource allocation* via a microeconomic architecture, whereby caches compete with each other for the right to execute a job submitted by a user, described in egg shell. Egg manages the bidding process and determines a winner. In the short-term, the auction process provides a dynamic and robust resource allocation mechanism with prices

[7]For reference, a typical ATLAS installation consists of 1233 Pacman scripts such as the one above, with 20687 lines of egg shell in total and deploys about 4GB of software. Most of the ATLAS egg shells are produced automatically from their build system. Parts, however, are hand written. Pacman typically creates more than 1000 new installations per day around the world, with more than 800,000 downloads of Pacman to more than 50 countries as of March 12, 2006.

linked to resource demand. In the long term, prices provide signals to guide future investment in resources and allow for accounting by parties such as funding agencies. All local schedulers and bidding algorithms employed by caches are fully extensible to allow for continual improvement.

The first element in the microeconomic architecture is a *bidding language* that allows a user to state a willingness-to-pay. Egg does not require a user to state resource requirements explicitly. For example, a Physicist interested in running computational analysis on a local grid does not need to estimate the length of time, or file space, that the computational process requires. The responsibility for resource estimation is pushed to caches: a cache needs an estimate of resource requirements to determine its opportunity cost for accepting a job.

Egg supports an expressive language to allow users to describe a tradeoff between completion time and value. A user can bid a *price schedule*, i.e. a willingness-to-pay for different job completion times. For simplicity we initially support a linear price schedule. For example, a typical bid defines (`@10.Harvard.Alice,@2.Harvard.Alice,Apr-01-06 00:00:01`), which describes a monotonically decreasing willingness-to-pay of $2 + (10 - 2)(t - t_0)/(t_d - t_0)$ eggs, where t is the time of completion, t_d is the maximal deadline (`Apr-01-06 00:00:01` in the example), and t_0 is the time at which the bid is submitted to Egg. A simple special case is a constant willingness-to-pay with a hard deadline. The bid is the *maximal* willingness-to-pay, the payment actually made by a user depends on the current balance of supply and demand. A user can also specify a *minimal reliability*, e.g. "I will only consider caches with reliability $\geq 99.9\%$." The Egg infrastructure maintains a *reliability* metric for caches, which is a measure of the frequency with which a cache has failed to meet a deadline.

Jobs are submitted (as egg shell) to multiple caches that satisfy the environment and reliability requirements specified in a job, and are willing to accept the currency specified by the user. Caches providing discovery services can facilitate this process of matching jobs with caches that are willing and able to generate bids. The job is ultimately allocated to the cache that responds with the lowest offer, and the cache receives this amount upon the successful completion of the job (and receives no payment otherwise.)

The microeconomic architecture carefully constrains this bidding process to provide *strategyproofness* to users: Egg guarantees that a user minimizes her payment and maximizes her chances of successful completion of the job if and only if she truthfully reports her willingness-to-pay and deadline considerations. Strategyproofness provides simplicity: Egg pro-

vides the benefits of an economic framework while hiding the potential complexities from users.

In achieving strategyproofness, while respecting the autonomy of caches, the most important architectural element is provided by *price tables*. Each cache represents its bid for jobs by populating entries in a price table. Loosely, a price table for cache i specifies a price $p_i(Q,t)$ for some quantity of resources Q allocated starting at time t. Egg requires monotonicity properties of prices in a price table, such that prices increase with larger quantities. Each cache must maintain prices up until a time horizon, some T time steps into the future. A cache can change entries in the price table, but can only *increase* its posted price $p_i(Q,t)$ for each (Q,t) pair. The Egg platform receives an estimate of compute resources $\hat{Q}$ from the cache, and then determines the bid from each cache by inspection of the price tables.

Caches retain flexibility to decide which kinds of jobs to schedule. For example, economically-motivated caches would set prices to maximize revenue given local knowledge about job characteristics and hardware characteristics.

5. Macroeconomic Architecture

The macroeconomic architecture in Egg is designed to support basic economic functions: the creation of currency, the security of currency, and currency exchange. Critically, and perhaps uniquely amongst current virtual economies, the Egg macroeconomy allows multiple currencies. This provides for complete autonomy with respect to policy to the many stakeholders on heterogeneous grids. Policy is then supported through the following mechanisms: (a) anyone can create a currency, "print" arbitrary quantities of the currency, and decide how to allocate currency to users; (b) an owner of compute or file servers can control access via restrictions on currency and identity, and by placing currency-specific limits on resource allocations.

Of course, just printing currency cannot make a user rich: currency only has value if it can be spent at caches, or exchanged for other currencies with spending power. On the other hand, anyone can be an Alan Greenspan (Bernard Bernanke?!) for their own economy. Every currency requires a bank, which is responsible for maintaining the security of the account of any user holding the currency, and also to provide important services such currency exchange. Moreover, every user has her own bank (even if she does not generate her own currency). These banks are involved in the transfer of currency between users.

Egg provides a secure identity infrastructure. For instance, the Harvard bank can certify its identity by the use of cryptographic signatures. When currency changes hands it is signed to indicate the parties involved in the exchange. For example, if Harvard generates 10 eggs and gives them to Laura then Laura's bank holds `@10.Harvard.Laura` eggs, indicating this transfer. The Harvard bank also keeps a record that Laura has 10 Harvard eggs. Laura can now grant `@5.Harvard.Laura` eggs to Saul and Saul's bank would hold `@5.Harvard.Laura.Saul` eggs. Saul's bank could also contact the Harvard bank, as the "bank of record" for Harvard eggs, and claim possession of these eggs at which point the Harvard bank would update its record.[8]

In Egg, caches can control access by specifying which currencies are accepted. For example, consider a simple computer cache which attempts to execute any egg shell given to it. If the cache accepts `@*.Saul` this is just as secure as putting Saul's public key in `.ssh/authorized_keys` (since Egg uses OpenSSL cryptography) with the substantial additional merit of having accounting, a chain of authorization if "`*`" is not the empty list, and a chance to pre-examine and record whatever Saul executes on the system. Similarly, if the computer accepts `@Harvard.*`, anyone holding Harvard currency can compute. In a more restrictive configuration with the computer accepting `@Harvard.Laura.?`, anyone who Laura gives Harvard currency to directly has access to the computer.

Caches which earn currency and which bid on incoming jobs will usually also have a *home currency* used as the unit of resource estimation and bidding. In such cases, the cache may earn accepted foreign currencies by using a bank to establish and exchange rate with the home currency.

In the global economy, it is partly through the control of monetary supply that countries can implement socio-economic policy and the same is true for the Egg economy. By allowing multiple currencies, a computational grid created by the Chinese government can interoperate with a computational grid created by the US government, but without either party ceding control of policy. China does not need to worry that the US bank will flood the economy with a surfeit of US eggs because users in China can hold China

[8]Thus, the Egg currency is not formally a bearer currency in the sense that a recipient bank cannot *independently* establish that the validity of eggs. But, we achieve good decentralization in practice, e.g. for transfers between mutually trusted banks. If Saul's bank is concerned about the security of Laura's bank (e.g. perhaps Laura's bank also gave 10 eggs to Margo), then in performing this transaction Saul's bank can first check with Harvard's bank that the eggs are still Laura's to transfer.

eggs, and the exchange rate would move against the US currency.

In a globe-spanning physics research project like ATLAS, the Egg platform would allow physicists to effectively allocate globally available resources directly in the natural terms of the research. Suppose that a high level policy decision concludes that analysis searching for the Higgs boson over the next month is so important that it requires 70% of the global ATLAS resources. The Egg currency allows a policymaker to specify this without making decisions on details such as who has access to which computers, storage and network resources, and with what priorities for what periods of time. By further delegation, someone managing the Higgs analysis effort can then provide additional refinement, e.g. specifying the fraction of resources are used for Monte Carlo simulation compared with analysis, or which persons or groups get to spend the currency. At the same time, owners of resources can maintain complete control over who has access to their systems and whether they contribute to the Higgs effort within ATLAS.

Note that the macroeconomic functions described here do not enter into consideration of the actors in the system. Users can organize resources that they have access to and can express their intentions in the currency units that they are familiar with when necessary.[9] Cache owners will simply list which currencies are accepted. Managers of organizations generate and transfer the currency that they are used to depending on needs and priorities.

Banks, like everything else in Egg, are also caches and autonomous agents in their own right and able to pursue local policies in establishing exchange rates. We intend to perform simulations to better understand the effect of various methods to compute exchange rates and perform exchange, both to understand robustness to shocks and also to determine which monetary metrics need to be instrumented by the Egg platform (e.g. real currency supply, inflation, etc.).

6. Closing Comments

Egg provides an extensible and economics-inspired open grid computing platform. This is a multi-year effort involving close collaboration between

[9] A user with `@BU` (Boston University) eggs can still state willingness-to-pay in BU eggs. Banks are used to generate quotes for currency exchange and enable competition across domains. Suppose a Harvard cache wins. On successful completion of the job, the Harvard bank would perform currency exchange from `@BU` into `@Harvard` eggs, credit these to the Harvard cache, and finally debit `@BU` eggs from the user's account. The net effect is that the Harvard bank is holding some `@BU` eggs.

computer scientists, computational physicists, and economists. Egg is spawning many subprojects. For instance: (a) statistical machine learning to predict resource requirements; (b) methods of computational mechanism design for sequential and strategyproof resource allocation; (c) opportunity-cost based schedulers; (d) algorithms to compute exchange rates; (e) languages for environment computing. Vital to Egg's success as a platform is its extensibility and openness: our current focus is on defining and implementing the Egg platform together with initial versions of various caches that we find useful. Continual innovation will ultimately provide for sustainable and successful grid computing.

6.1. *Related work*

The Globus Toolkit[10] and Condor [12] provide the current *de facto* architecture for resource management in grid computing. However, as argued by Foster et al. [9], grid computing is "brawn" without "brain." We'd go further and argue that grid computing is just too cumbersome and complex, and with insufficient kinds of expressiveness (for both policy and use), to approach Internet scale. Also, while Foster et al. [9] suggest the use of agent technology for the automatic creation of "virtual organizations" [6], our view is that stakeholders– funding agencies, Deans of engineering schools, managers, etc. –should be provided with mechanisms to implement policy while agents put to use to price resources, predict job characteristics, adjust exchange rates, and other such well-defined tasks. Dumitrescu et al. [5] provide an alternate vision of policy for computational grids.

Many papers have proposed using market-based methods for resource allocation in networked computing [2,11,15,8,16,14], some of which has focused specifically on computational grids and federated systems [13,1,18,17,10]. However, the combined microeconomic and macroeconomic architecture, coupled with attention to policy and the need for decision autonomy differentiates Egg from these earlier works. To give a couple of examples, we are not aware of any work that allows for multiple currencies and considers macroeconomic issues such as exchange rates, nor are we area of any work that provides for strategyproofness to users (i.e. non-manipulability) despite the dynamics of grid computing environments and while still supporting seller autonomy in setting local price policies.

The microeconomic architecture adopted in Egg is inspired by recent

[10]http://www.globus.org

theories on the design of strategyproof allocation mechanisms in dynamic environments [7]. The macroeconomic design shares some of the goals expressed in the work of Irwin et al. [4], for instance in recognizing the importance that currency schemes support policy and allow delegation of resource access rights. Clark et al. [3] have written at length about the success of the Internet as a network platform, especially about the role of openness and end-to-end arguments in the support of continual innovation.

Acknowledgments

This work is partially supported by NSF ITR 0427348.

References

1. R. Buyya, D. Abramson, , and J. Giddy. NimrodG: An architecture of a resource management and scheduling system in a global computational grid. In *In Proceedings of the 4th International Conference on High Performance Computing in Asia-Pacific Region*, pages 283–289, May 2000.
2. Brent N. Chun, Philip Buonadonna, Alvin AuYoung, Chaki Ng, David C. Parkes, Jeffrey Shneidman, Alex C. Snoeren, and Amin Vahdat. Mirage: A microeconomic resource allocation system for sensornet testbeds. In *Proceedings of 2nd IEEE Workshop on Embedded Networked Sensors (EmNetsII)*, 2005.
3. David D. Clark, John Wroclawski, Karen R. Sollins, and Robert Braden. Tussle in cyberspace: Defining tomorrow's Internet. In *Proc. ACM SIGCOMM*, 2002.
4. D.Irwin, J.Chase, L.Grit, and A.Yumerefendi. Self-recharging virtual currency. In *Workshop on Economics of Peer-to-Peer Systems*, 2005.
5. Catalin L. Dumitrescu, Michael Wilde, and Ian Foster. A model for usage policy-based resource allocation in grids. In *In PolicyWorkshop2k5*, 2005.
6. I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the grid: Enabling scalable virtual organizations. *International Journal of SuperComputer Applications*, 15(3), 2001.
7. Mohammad T. Hajiaghayi, Robert Kleinberg, Mohammad Mahdian, and David C. Parkes. Online auctions with re-usable goods. In *Proc. ACM Conf. on Electronic Commerce*, pages 165–174, 2005.
8. I.E.Sutherland. A futures market in computer time. *Communications of the ACM*, 11:449–451, 1968.
9. I.Foster, N.R.Jennings, and C.Kesselman. Brain meets brawn: Why grid and agents need each other. In *Proc. 3rd Int. Conf. on Auton. Agents and Multi-Agent Systems (AAMAS)*, 2004.
10. K. Czajkowski and I. Foster and C. Kesselman and N. Karonis and S. Martin and W. Smith and and S. Tuecke. A resource management architecture for metacomputing systems. In *Workshop on Job Scheduling Strategies for Parallel Processing*, 1998.

11. K. Lai, B. Huberman, and L. Fine. Tycoon: A distributed market-based resource allocation system. Technical Report cs.DC/0404013, Hewlett Packard, 2005.
12. M. Litzkow, M. Livny, , and M. Mutka. Condor - A Hunter of Idle Workstations. In *Proceedings of the 8th International Conference of Distributed Computing Systems*, pages 104–111, June 1988.
13. M.Balazinska, H.Balakrishnan, and M.Stonebraker. Contract-based load management in federated distributed systems. In *First Symp. on Networked Systems Design and Impl. (NSDI)*, 2004.
14. I. Stoica, H. Abdel-Wahab, and A. Pothen. A microeconomic scheduler for parallel computers. In *Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing*, pages 200–218. Springer-Verlag, 1995.
15. M Stonebraker, R Devine, M Kornacker, W Litwin, A Pfeffer, A Sah, and C Staelin. An economic paradigm for query processing and data migration in Mariposa. In *Proc. 3rd Int. Conf. on Parallel and Distributed Information Systems*, pages 58–67, 1994.
16. Carl A Waldspurger, Tad Hogg, Bernado Huberman, Jeffrey O Kephart, and W. Scott Stornetta. Spawn: A distributed computational economy. *IEEE Trans. on Software Engineering*, 18:103–117, 1992.
17. R. Wolski, J. Brevik, J. Plank, , and T. Bryan. Grid resource allocation and control using computational economies. In *In Grid Computing: Making the Global Infrastructure a Reality*, pages 747–772. Wiley and Sons, 2003.
18. R. Wolski, J. Plank, J. Brevik, and T. Bryan. Analyzing market-based resource allocation strategies for the computational grid. *The International Journal of High Performance Computing Applications*, pages 258–281, 2001.