

RUR: A Proposed Model for CPU Marketing

Paul Cockshott[†]
wpc@dcs.gla.ac.uk

Viktor Yarmolenko[†]
v.yarmolenko@dcs.gla.ac.uk

Lewis Mackenzie[†]
lewis@dcs.gla.ac.uk

[†]Department of Computing Science, University of Glasgow, Glasgow G12 8QQ, United Kingdom

ABSTRACT

This paper describes a model for a marketing of CPU Cycles on the Grid. The model resolves the issues like: How do you find computers *willing* to do your job; How is payment made for the work done; How can proof be produced that the work has been done.

Categories and Subject Descriptors

D.4.7 [Operating Systems]: Organization and Design—*Distributed systems, Interactive systems*; J.7 [Computer Applications]: Computers in Other Systems—*Consumer products, Process control*; K.1 [Computing Milieux]: The Computer Industry—*Markets*; K.4.4 [Computers and Society]: Electronic Commerce—*Cybercash, Digital Cash, Distributed Commercial Transactions, Payment schemes*

General Terms

Design, Economics, Theory

Keywords

CPU Share, CPU Marketing, Distributed Computing, CPU Trading Model

1. INTRODUCTION

Existing Grid [1] protocols have been developed in the context of government or academic computing centres. In such centres access is usually based on quotas of computer time, allowing certified users to access super-computer resources. The characteristic of computer centres is that most of the jobs are based on batch processing, which diminishes the greatest potential of a Grid, that is dynamic resource discovery. Another characteristic of computer centres is that they are typically behind firewalls that are politically hard to reconfigure. This tends to limit Grid protocols to a few ports that are guaranteed to be visible through the firewalls.

The spread of broadband Internet over the last 3 years means that a huge new computing resource has come onto the In-

ternet. In terms of CPU power, this resource is substantially greater than the resources of super-computer centres. The processors in home machines are competitive, exist in large numbers, and now have communication links that make them available for a sufficiently long time. This has allowed the proliferation of peer-to-peer computing protocols for file sharing, instant messaging, VoIP, Video Streaming, *etc.* These protocols offer an immediate benefit to the users, are easy to set up and require no heavyweight certification.

There are obvious opportunities here for alternative protocols for distributed processing inspired by the peer-to-peer networks [27, 12]. Protocols which allowed home users to sell their idle computing time would call forth more resources than can be tapped using the current Grid resource. Traditional mechanisms [26, 19] for distributed CPU management are too rigid and static or do not include private individuals and small businesses for historic reasons.

In recent years, various attempts have been made to exploit such remote resources, but they were either too specific on the code level [10], application level [29], had some security issues [2, 15, 7], or were not scalable well on heterogeneous networks [2, 17]. In the SETI@home project [10] Berkeley's researchers realised the potential of home CPU power. Their code had to be manually installed and executed as a screen saver by users on voluntary basis. They showed the advantage of using home CPUs (currently getting 15 TeraFLOPs) to the most powerful computer, IBM's ASCI White (rated at 12 TeraFLOPs and priced \$110 million). SETI@home recently broke a record of largest computation ever performed [3]. Other researchers followed the suit: Life map modelling [31], cancer research [5], Clinical Trial Data [9], genome analysis [13] and protein folding [30], financial/economic modelling [11], cryptography challenge [4]. Others progressed further in that the screen saver code did not consist of a dedicated scientific application, but new task could be uploaded [21, 14]. Its limitations were non dynamic scheduling, not Grid enabled, tasks could not spawn other tasks and could not communicate with each other, job submission was centralised.

Currently they are working on value-for-value (v4v) scheme [25, 33] similar to the CX Project [23]. The latter is more flexible and probably the closest in addressing the problems outlined in this paper. CX is a Jini-Based Computation Exchange system in which arbitrary java tasks are submitted by clients to a market where they are picked up to be pro-

cessed. It uses eager task scheduling [6], but still does not get completely away from centralised job spawning approach. Other distributed systems are limited in their scalability or flexibility (MPI [2], RMI [15], PVM [7], DSM, *etc.*) security (CMCP[34], Cilk[18]) or did not include v4v (virtually all).

We have found no report of a system or infrastructure, which would offer a credible yet flexible service of job-done-checking along with secure payments/credits scheme. Attempts to tackle these issues so far were non-generic or tackled part of the issue [29, 21, 24]. The review above suggests an increasing need for a standardized mechanism for v4v computing. This paper proposes a system, RUR [8] to create a market for idle computer cycles.

2. STATING THE PROBLEM

As was mentioned earlier, the protocols which allowed home users to sell their idle computing time easily are the next logical step to popularization of the distributed computing and hence making huge resources available to a wider range of users.

There are several good reasons for this. First of all, the development of a market in CPU time would have wide economic implications. It would open up a new area to commercialisation in a manner analogous to the creation of the Web. One can envisage that it would result in the formation of new forms of companies specialising to meet this market. Just as the web created a market for companies supplying server farms, one can envisage similar opportunism opening up to create commercial processor farms. The likely initial beneficiaries of this would be the firms who currently operate server farms.

Secondly, the additional network traffic created by the distribution of processing resources would be to the benefit of Telecoms companies and ISPs.

Thirdly, there would be market, which would operate in a fashion similar to banks in the clearing of credit and debit balances for processing work done, and in converting these balances into *real* money. It would be reasonable to expect that other companies would come into existence specialising in the provision of intermediary services - bulk discounted processing time and the like.

Finally, firms which have occasional bursty requirements for massive processing power - for example the film industry who need large quantities of render time for a particular film - would be saved the capital overhead of having to buy farms of servers themselves.

However, any proposal to trade CPU cycles between complete strangers inevitably encounters the following problems:

1. How can one deal with the heterogeneity of computers that exist on the Internet, allowing a job to be specified without any prior knowledge of the operating system or processor on which it will run.
2. How can the person offering CPU cycles be sure that no malicious actions will be performed by the program that runs on their computer.

3. How can a system of accounting and payment be established for the work done.

We believe that the first two problems can be solved by a modest extension of existing Java technology. Java already provides a compile once, run anywhere, paradigm through its portable byte-codes [20]. Its incorporation into web browsers means that running Java code on client machines is commonplace. We are focusing our investigations on purely CPU-based parallelism, that is, we are assuming that tasks running on a client machine will not be able to access the local file store. All data needed for the computation will be sent over the Internet from the machine originating the jobs. All results will be sent via memory-to-memory transfer, without accessing local file store. We can therefore concentrate on solving the question of accounting, payment and audit trails for compute tasks. The remaining problems in hiring out computer time on an anonymous market are:

1. How do you find computers *willing* to do your job.
2. How is payment made for work done.
3. How can proof be produced that the work has been done.

We propose to use a peer-to-peer protocol with servers analogous to systems like Gnutella [27]. Machines are divided along Gnutella lines into two categories labour exchanges (LabEx) and server entities (ServEnts). ServEnt can both initiate and accept compute jobs on the network. The entities involved in our model are Abstract Juridical Personalities, (AJP), LabEx and ServEnt. An AJP is either a person, a company or any other institution capable of pursuing a claim in a court of law. A LabEx is a computer with linkage to the banking system capable of triggering payments between AJPs. A ServEnt is a computer owned by an AJP that is made available to trade CPU cycles. It acts as a Robot for hire. The term Robot which entered English from the Czech play RUR [8], had a prior meaning, among other concepts relating to work, of labour services owed by a peasant to a feudal lord or Boyar. Rossum's Robots were his mechanical slaves. Our computers stand in a similar relationship to us.

To deal with problems 1 and 2 we use the concept of Labour Exchange. These embody both the administrative notion of a labour exchange in the British social security system as a place to which you report to find labour when you are unemployed, and also Robert Owen's idea of labour exchanges as places where citizens' mutual labour debts would be canceled [22]. These notions derived from public policy are given physical embodiment in a server similar to those used in the Gnutella network. Since the computational tasks will be distinct and run on computers of different powers with different instruction sets we need some means of equating work on these various machines. This is analogous to the problem of reducing complex to simple labour in value theory. We propose to quote all job costs in standard Java CPU minutes, equating the performance of different machines using a representative Java instruction mix. The costs in Java CPU minutes would be converted to money prior to final settlement using a Monetary Equivalent of Computer Labour.

To deal with problem 3 we use notions from Turing [32] and from public key cryptography [28]. Turing posited both a model of computation, the Turing Machine, and a generalisation, the so-called *Universal Turing Machine* (UTM). All Turing Machines are finite automata augmented by one or more infinite read/write sequential stores or *tapes*. The UTM is one which when given an input tape made up of the sequence $u_i p$ will emulate the behaviour of any other Turing machine i applied to input data p . The Java Virtual Machine is computationally equivalent to a bounded UTM. When provided with an input consisting of a jar [] file u_j followed by a sequential data stream p it will run the j -th possible Java program on data p . The set of possible Java programs can be enumerated by treating their jar file encodings as binary integers. Since any finite bounded Turing Machine can be emulated as a Java program working on an instance of the Vector class, it follows $\forall u_i \exists u_j$.

Suppose we run a specific Turing Machine u_i on input p_k then, provided the machine halts, we are left with an answer a_k that was fully specified by $u_i p_k$. The answer was implicit in the specification of the problem but computational work was required to produce it. It is this computational work that we propose to trade. Suppose that when machine A agrees to do work for machine B , they record a sealed version of $u_i p_k$ with a trusted third party E . When A completes the task and sends a_k to B , it also sends a copy to E . A has then registered proof with E that the contracted computational task has been done. If B disputes payment, the sealed copy of $u_i p_k$ can be run by E on a reference UTM to see if the answer is a_k .

It is common usage in file sharing networks to use secure hashes to identify files on servers in a way that is independent of their file names [16]. A similar approach can be used to the pairs $u_i p_k$ and the answers a_k , so that instead of recording the entire program, data and answer at the server, a secure hash of the data, program and answer are stored. With sufficiently long hashes the chances of misidentifying programs data and answers are vanishingly small.

We believe that within the context described above, proof of execution of a task is relatively easily demonstrated. Providing secure protocols to prove the performance of tasks is not trivial in case of remote processes sub-contracting part of their work elsewhere. It then becomes necessary to fix financial responsibility for the failure of a distributed task on the defaulting sub-contractor machines.

3. PROTOCOLS

We now present a set of protocols to implement this conceptual model. In this section we describe a unit of labour and tasks that must be performed by LabEx and ServEnt. In 4 we describe general messages to be passed between various entities.

3.1 Tasks of LabEx

1. To keep a record of currently available ServEnts and their IP addresses.
2. To record the existence of AJPs and associate each active ServEnt with a unique AJP.

3. To maintain an accounting system recording the number of labour credits performed by or done for ServEnts belonging to each AJP.
4. To allow the purchase of labour credits by AJPs for real money.
5. To allow payment to AJPs for the labour done by their ServEnts.
6. To allow each ServEnt to have a current charging rate for labour it is willing to do.
7. To provide each ServEnt with a job to do with a list of the ServEnts who are willing to do the job at the lowest price.
8. To communicate with other labour exchanges to reconcile accounts and communicate the availability of ServEnts.

3.2 Tasks of ServEnts

1. To record its availability for hire with a LabEx along with the AJP for whom it works.
2. To notify the LabEx of its IP address.
3. To notify the LabEx of the wage rate for which it will work.
4. To notify the LabEx of each contract of work that the ServEnt accepts, and of when that contract has been completed.
5. To periodically ping the LabEx notifying the LabEx of its continued availability.
6. To unconditionally accept tasks from its master.
7. If the task is too onerous to hire assistance from other ServEnts for this task.
8. When idle, to accept work contracts from other ServEnts.

3.3 Labour Unit

All work is measured in Nominal Java Minutes. This will be the time that a job takes on some reference machine. A benchmark routine would be provided that ServEnts can use to rate the speed of their CPU. The routine will return the number of seconds that a 1 minute job will take on the current CPU.

Charging rates may be quoted by ServEnts taking advantage of their CPU and their master's generosity. Thus a ServEnt on a fast CPU might say that it would charge only 20 nominal seconds for a 1 minute job. Or a user who did not want to be paid, might say they would charge 0 seconds for a 1 min job.

4. MESSAGES

A master wishing to higher out his computational services starts up a registration Java application on their machine. This synthesises a cryptographic key pair and informs a LabEx machine of the public key. In return the LabEx sends the registration application an AJP's ID - a unique sequence of digits. As a matter of policy the LabEx can at this point set a labour overdraft limit associated with the master. This sets up a master's account. A master who wishes to pay externally verified money for work done on their behalf or to earn money for work done by their ServEnts, can register an external bank or credit card account with the LabEx. Messages requesting these are signed with the master's private key. The LabEx may at this point alter the overdraft limit of the master.

A master may order the purchase of work credits from the LabEx. It also can enquire the labour credit balance. All messages are signed by a private key. Next follow the messages from ServEnts exchanged at different stages.

4.1 Register Presence

	ServEnt		LabEx
HereIAm (Master: AJP's ID, signature)	→		
		←	YouAre(SID)

This message exchange sets up a ServEnt's ID (SID) for the ServEnt and allows the LabEx to associate this with an AJP. An SID, can be, for example, a pair consisting of an IP address:port and a unique decimal integer generated by the LabEx.

4.2 Charging Rate

	ServEnt		LabEx
Icharge(xsecs, SID)	→		

The charging rate states how many seconds the ServEnt charges for a nominal 1 minute job.

4.3 Finding ServEnts

	ServEnt		LabEx
WhoIsAvailable (number wanted)	→		
		←	They Are (ServEnt - list)

LabEx sends a list of available ServEnts. These should be the cheapest ones currently free. Each element in the list is a pair: SID and the charge rate.

4.4 Contracts

The following message interchanges are involved in setting up contracts under which a machine undertakes to do work for another owner. A contract consists of the following:

Requesting ServEnt : SID2
Target : SID1
Nominal seconds : number

Charged seconds : number

Job ID : hash code

Signature : signed with a private key

An accepted contract is a contract counter signed with the private key of the accepting party. A job ID, is the hash code of the job details. The job details consists of a serialized jar file followed by a serialized file of parameters. A rescode is the hash code of the answer. The answer is a stream of bytes.

Earlier we had said that verification of fulfillment of a contract is by sending the program and answer to a trusted 3rd party. We now substitute secure hashes of the program and answer being sent to the LabEx. This serves the same effect as sending the originals but saves bandwidth. Provided the hash is long enough the probability of falsely registering completion is vanishingly small.

4.4.1 Messages from ServEnt to LabEx

	ServEnt		LabEx
CheckCredit(Contract)	→		
		←	CreditIs(good/bad)
Accepted(Accepted, Contract)	→		
IveDoneIt(Contract,rescode)	→		
YesHeDidIt(Contract,rescode)	→		
		←	PaymentMade (Contract)

4.4.2 Messages between ServEnts 1 and 2

	SID2		SID1
WillYouDoIt(Contract)	→		
		←	YesIWill (Accepted, Contract)
			or
		←	NoIWont(Contract)
HeresTheJob(Contract, Jobdetails)	→		
		←	Here'sTheAnswer (answer, rescode)

The job setup protocol is:

1. SID2 contacts SID1 with a *WillYouDoIt* message.
2. SID1 contacts the LabEx with a *CheckCredit* message. If this is bad it sends a *NoIWont* message back to SID2, otherwise, if SID1 is free, it responds by sending a *YesIWill* message to SID1 and an *Accepted* message to the LabEx.
3. SID2 responds by sending the *Jobdetails* message.
4. SID1 runs the job.
5. SID1 computes the hash of the answer and sends the answer to SID2.
6. SID1 sends the hash of the answer to the LabEx as proof of having done the job.
7. SID2 then sends *YesHeDidIt* message to the LabEx.

8. At this point the LabEx credits the account of SID1's master the cost of the job and debits the account of SID2's master with the cost of the job plus overheads.
9. The LabEx then sends SID1 a *PaymentMade* message.

If SID2 denies that the job was completed, the LabEx can run a test check on the hash code of the answer, asking for a copy of the original program and data. The LabEx can verify that this was the original program and data by checking against the hash code registered with the LabEx. The answer obtained is then tested against the hash code of the answer sent by SID1.

If SID2 is repeatedly seen by the LabEx to fail to acknowledge work done by multiple ServEnts then it will be rated as bad by the LabEx. If a ServEnt finds that other ServEnts are repeatedly failing to make payment, then it can unilaterally refuse to accept further work from them.

5. CONCLUSION

We have presented a model of distributed computation in which payment is made for computing work done. The protocols support verifiable contracts with an audit trail of the work done. We believe that the adoption of this or some similar mechanism is crucial to the creation of a viable market in CPU cycles.

6. REFERENCES

- [1] Global grid forum (<http://www.ggf.org>).
- [2] Mpi: A message-passing interface standard. In *Message Passing Interface Forum*, May 1994.
- [3] Bbc news online: Sci/tech. tuesday, august 17. In <http://news.bbc.co.uk/1/low/sci/tech/423022.stm>, 1999.
- [4] Distributed computing puts the net to work. In *Washington Post*. Distributed.net, November 1999.
- [5] Compute against cancer. In <http://www.computeagainstcancer.org/>, 2003.
- [6] Z. K. P. W. A. Baratloo, M. Karaul. Charlotte: Metacomputing on the web. In *Proceedings of the 9th Conference on Parallel and Distributed Computing Systems*, 1996.
- [7] J. D. W. J. R. M. V. S. S. Al Geist, Adam Beguelin. *PVM: Parallel Virtual Machine - A Users' Guide and Tutorial for Networked Parallel Computing (Scientific & Engineering Computation)*. MIT Press, 1995.
- [8] K. Capek. R.U.R. In *Four Plays*. Methuen Drama, July 1999.
- [9] J. M. Cummins. Overview of the companys research and development projects. In *Bio Venture Conference & Symposium*. Amarillo Biosciences Incorporated (<http://www.amarbio.com/>), December 2003.
- [10] C. K. D. Gedye. Seti@home. 5th International Conference in Bioastronomy, July 1996.
- [11] E. B. *et al.* 'complexity science' employed to develop winning business strategies. In *News Release: Fairfax, VA*. Icosystem Corporation, Cambridge, MA 02138, August 2001.
- [12] N. M. *et al.* *Peer-to-Peer: Harnessing the Power of Disruptive Technologies*. O'Reilly, 2001.
- [13] T. J. O. *et al.* Aids@home project (<http://fightaidsathome.scripps.edu/>). Olson Laboratory at The Scripps Research Institute.
- [14] W. L. G. *et al.* Accelerating scientific discovery through computation and visualization ii. *NIST Journal of Research*, 107(3):223, 2002.
- [15] W. Grosso. *Java RMI*. O'Reilly, 2001.
- [16] B. W. T. W. H. Ian Clarke, Oskar Sandberg. Freenet: A distributed anonymous information storage and retrieval system. In *Designing Privacy Enhancing Technologies: International Workshop on Design Issues in Anonymity and Unobservability, LNCS 2009*, ed. by H. Federrath. Springer: New York, 2001.
- [17] R. L. J. Nielocha, R. Harrison. Global arrays: A portable shared memory programming model for distributed memory computers. In *Supercomputing '94 Proceedings*, page 340, 1994.
- [18] C. F. Joerg. The Cilk system for parallel multithreaded computing. In *Ph. D. Thesis*. MIT Dep-t. of EECS, January 1996.
- [19] E. G. C. Jr. *Computer and Job Shop Scheduling Theory*. Wiley, New York, 1976.
- [20] D. Kramer. The javatm platform: A white paper. Sun Microsystems, May 1996.
- [21] D. McCullagh. P2p fans predict rebound. In *News Release: Wired News* (<http://www.wired.com/news/business/0,1367,48231,00.html>). Parabon Computation (<http://www.parabon.com/>), November 2001.
- [22] R. Owen. A new view of society and report to the county of Lanark. Penguin, May 1996 (first published (1813)).
- [23] D. M. P. Cappello. A scalable, robust network for parallel computing. In *Joint ACM JavaGrande - ISCOPE 2001 Conference (Stanford University, California)*, pages 78–86, June 2001.
- [24] B. C. K. C. E. L. K. H. R. Y. Z. R. D. Blumofe, C. F. Joerg. Cilk: An efficient multithreaded runtime system. In *Proc. 5th ACM Symposium on Principles and Practice of Parallel Programming*, July 1995.
- [25] M. A. B. R. F. Brooks, I. N. Lings. *The Service Industries Journal*, London, October 1999.
- [26] J. K. L. A. H. G. R. K. R. L. Graham, E. L. Lawler. Optimization and approximation in deterministic sequencing and scheduling: A survey. *Annals of Discrete Mathematics*, 5:287–326, 1979.

- [27] M. Ripeanu. Peer-to-peer architecture case study: Gnutella network). In *Technical Report*. University of Chicago, 2001.
- [28] R. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120, February 1978.
- [29] C. Sharp. Business integration for games: An introduction to online games and e-business infrastructure. In *White Paper*. IBM, 2003.
- [30] M. Shirts and V. S. Pande. Screen savers of the world unite! *Science*, 290(5498):1903–1904, December 2000.
- [31] P. Tooby. Lifemapper screensaver produces species distribution maps and models. In *News about the NPACI and SDSC Community* (<http://www.npaci.edu/online/v6.14/lifemapper.html>), volume 6. NPACI & SDSC, July 2002.
- [32] A. Turing. On computable numbers, with an application to the entscheidungsproblem. In *Proceedings of the London Mathematical Society, Ser. 2*, volume 42, 1937.
- [33] R. Van Haastrecht and M. Bekkers. In *24th EMAC Conference Proceedings*, page 1243, 1995.
- [34] P. Welch. CSP networking for Java (jcsp.net). In *Global and Collaborative Computing Workshop, ICCS*, April 2002.