

# Implementing fine and coarse grained payment mechanisms using WebCom

Adarsh Patil, Christoffer Norvik, David A. Power and John P. Morrison  
Department of Computer Science,  
University College Cork,  
Ireland.  
{adarsh,c.norvik,d.power,j.morrison}@cs.ucc.ie

## Abstract

In a Grid environment applications are executed on distributed resources. Applications can be divided into separate parts for distributed and sequential execution. Typically, complete applications consisting of many parts may be executed on both local and remote resources. Computing resources may expect some form of payment in exchange for services used. In this situation there are two scenarios where credit can be claimed: either in increments by executing parts of an application (fine grained) or by executing a complete application (coarse grained). This paper presents fine and coarse grained Condensed Graph applications and methods adopted for secure submission of applications. It also contrasts the payment mechanism for these approaches in WebCom. Many grid middlewares such as PBS, Torque, LSF and others employ a coarse grained approach for application distribution. Fine grained execution is partially supported by subdividing the application. Here we show both fine grain and coarse grain approaches to application distribution.

*Keywords: Grid Economy, Trust Management, Keynote, Hash Chain Coins, Fine grained payment, Coarse grained payment, Condensed Graph*

## I. INTRODUCTION

Grid environments consist of valuable resources spread across different interconnected domains. This provides a platform for large scale distributed computing, collaborative computing and service oriented computing. This environment consists of a variety of resources such as PCs, workstations, supercomputers, clusters, mainframes and special instruments. Researchers and end users belong to a different domain of interests capable of executing applications and investigating their execution characteristics. Physical resources of varying performance are autonomous. Some of these have potential to process simple fine grained tasks of an application while others have the potential to process the complete application.

Grid middlewares provide a glue for connecting Grid resources. These support various services such as accounting, queueing, scheduling, application composition, resource and job management, security, uniform access and charging. Currently, some middlewares provide these services at application and task/job level. However, most of them only support application level scheduling and not fine grain task/instruction level scheduling.

A coarse grained application consists of smaller fine grained tasks, each of which might need a particular resource for its execution. Job Management systems [20] are employed when scheduling tasks to the resources. Most of the job management systems guarantee coarse grained application execution through their queueing system.

Accounting and charging procedures have been implemented in some job management systems. These methods are generally applied to complete applications rather than to fine grained constituent tasks. Recently, some accounting and charging [15] [29] methods such as GridBank [9], "Virtual Users" at Polish National Cluster [3], Template Accounts [16], DGAS [6] and GRACE [13] have been the focus

of research. In this article, we explain fine grained and coarse grained (complete) application execution using WebCom. This paper also contrasts on the payment behaviors of both of these methods.

The rest of this paper is organized as follows: Different payment techniques and Security mechanism are enumerated in Section II further discussion on these is outside the scope of this paper. WebCom and its application execution is discussed in Section III. Section IV presents the techniques used for both coarse and fine grained application execution. Section V describes payment for fine grained computing using WebCom. Section VI describes payment for coarse grained computing using WebCom. Section VII briefly outlines the workflow involved. Section VIII contrasts the two payment mechanism. Section IX describes the experimental test-bed, procedures and preliminary results. Finally Section X presents some conclusions and future work.

## II. PAYMENT MECHANISMS AND SECURE CONNECTION EMPLOYED IN GRID ENVIRONMENT

A Grid environment which asserts to pay its clients for processing applications and its parts has to adopt to some payment mechanisms. The payments mechanisms are: Pay as you go, Direct Debit, Contract, Prepaid scheme, Quota Based. These can be implemented by using digital currency techniques such as: NetCheque [22], Mojo [1], NetCash [17], Paypal [8], Tokens [4] [28]. Typically, digital currency payment mechanisms use these security protocols for transactions and interaction. Some of these include Secure Socket Layer (SSL), Transport Layer Security (TLS) [7], Multilevel Security (MLS), Public Key Infrastructure(PKI) [27] [2], Kerberos [24] and Role Based Access Control(RBAC) [23] / Key Exchange.

## III. WEBCOM AND APPLICATION EXECUTION

WebCom [19] is a ‘fledging Grid Operating System’, designed to provide independent service access through inter-operability with existing middlewares. It is based on the Condensed Graph (CG) [18] model of Computing, which is a graph based model that uses Directed Acyclic Graphs (DAGs). The core architecture of WebCom (See Fig. 1) consists of following modules: an engine module, a distributor module, a fault tolerance module, a security module, communication/connection module, information module and a job manager module.

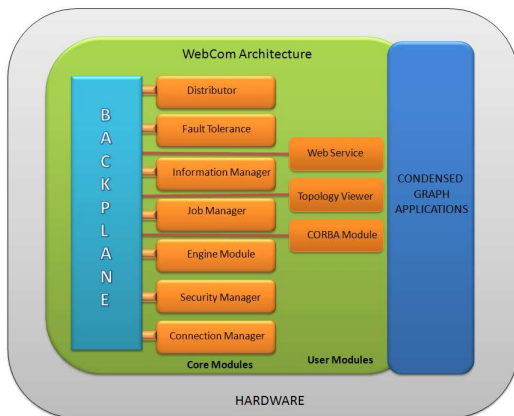


Fig. 1. The WebCom Architecture

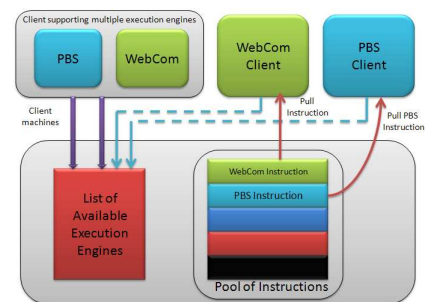


Fig. 2. The WebCom Distributor Module.

The Condensed Graph *Engine Module* (CGEngine), which executes applications expressed as condensed graphs by uncovering fireable instructions and placing them in a **Pool**. The *Distributor Module* performs the actions found in the traditional scheduler and load balancer. It receives instructions from the pool and selects a client to execute an instruction based on configured **Policies and Algorithms**. Once a node is selected for distribution, it is placed on to the clients server side allocation queue. The *Fault Tolerance module*, monitors client resources and executing applications and reschedules work that was sent to machines that failed. The *Security Module* can be used to enforce different security policies on

executing applications and user restriction/access. The *Communication/Connection manager module* is responsible for transporting messages to the selected WebComs. The *Information module* is responsible for providing the status of the resources and module information of each WebCom. The *Job Manager module* is responsible for tracking the execution of a job across the entire WebCom network.

WebCom separates the application and execution environments by providing both an execution platform and a development platform. Applications are specified as Condensed Graphs in a manner that is independent of the execution architecture. The independence provided by separating these two environments facilitates computation in heterogeneous environments; the same *CG* programs run without change on a range of implementation platforms, from silicon-based Field Programmable Gate Arrays [21] to the Java-based WebCom. Fault tolerance, load balancing, scheduling and exploitation of available parallelism are handled implicitly by WebCom without explicit programmer intervention. Application execution is initiated by a user on a single WebCom. As nodes become available for execution, they are encapsulated within messages and passed to the distributor module.

#### A. Distributor Module

The Distributor Fig. 2 makes decisions on when and where to distribute instructions. Its operation is dictated by policies supplied both by site owners (statically) and users (at application runtime). Policies specify the behavior of the Distributor module. These specify settings such as when to request work and what algorithms to use for load balancing, for example. Algorithms are the Java implementation of Load Balancing algorithms, e.g. Round Robin and FIFO. Users can supply their own implementation of algorithms which can be used by their policies.

Policies provide rules and heuristics that allow the Distributor to make scheduling, load balancing and communication decisions. The behavior of each WebCom instance is dictated by a hierarchy of policies. This hierarchy spans administration, system, graph and node policies. The site policy supersedes all others and is specified by the system owner. Next in the order of precedence is the administration policy, followed by graph and node policies. Graph policies travel between WebComs with their associated graphs. Node policies travel with associated nodes. Graph and node policies can be supplied by the user at run time.

Policies are specified as text files, and hence changing a policy is low impact, no code re-writing is needed. Policy changes can be carried out dynamically. Policy specifications can include heuristics such as pre-staging of data, node priorities and node groupings where, for example, inter-dependant nodes can be dynamically allocated to the same machine for execution (perhaps due to side-effects).

The Distributor Module has a pool of instructions handed to it by the backplane. It iterates through this pool examining each instruction and its associated policies to determine if it can be executed locally or remotely. Client WebComs will then pull those instructions allocated to them.

### IV. COARSE AND FINE GRAINED APPLICATION EXECUTION

WebCom expresses applications as Condensed Graphs (*CGs*). When expressing a problem as a Condensed Graph, nodes represent tasks and arcs determine the way these are sequenced for execution. By altering the connection topology of the graph, various evaluation orders can be specified. Sequencing constraints can be specified statically by a programmer but they can also be altered dynamically using feedback from the underlying execution environment.

Nodes in a Condensed Graph can represent fine grained tasks, complete applications or other condensed graphs. When a node representing a *CG*, known as a condensed node, is executed, the associated *CG* becomes an independent parallel subcomputation. This process is known as evaporation. An evaporated graph may contain further condensed nodes; in such a case, evaporation may occur recursively.

Once such a node has completed its operation, a result message is created and returned to the WebCom that the node was distributed from. This WebCom then incorporates the returned result into the node's graph and the execution proceeds. If a remotely distributed node fails to complete its task, the fault-tolerance module on the distributing WebCom will cause the node to be rescheduled to an alternative compatible WebCom. If no such WebCom is available, the node is retained for subsequent assignment.

### A. Bank

A Bank is the authority trusted by the Customer and WebCom ( both the server and the clients). The Trust Management *policy* allows the bank to issue contracts on the servers behalf. The Bank is responsible for creating a *contract* for the customer, enabling the customer to use specified services offered by WebCom. As discussed in [14], hash chain based micropayments [5] [12] [25] and Keynote Trust Management [11] [10] [26] are being used for the payment mechanism.

Trust Management is an approach to constructing and interpreting the trust relationships between public keys that are used to mediate security critical actions. Cryptographic credentials are used to specify delegation of authorization for services among the public keys. The Micropayment scheme is intended to support very low-value payments and operate as below.

A payer (Customer) generates fresh random seed  $s$ , and computes  $h^n(s)$ , where  $h()$  is a cryptographic one-way hash function. If  $s$  is known only to the payer, then  $[h^{n-1}(s), n-1, val] \dots [h^1(s), 1, val]$  provide ordered chain of micropayments, each one worth  $val$ . At the start, the payer provides the payee with  $[h_n(s), n, val]$ , which as a contract for  $(n-1)$  micropayments. A payee (WebCom server/connected WebCom clients) who has securely received  $i$  micropayments,  $[h^{n-1}(s), n-1, val] \dots [h^{n-i}(s), n-i, val]$ , can use the hash function  $h()$  to check their validity against the initial contract. Since  $h()$  is a one-way hash function it is not feasible for the payee to forge or compute the next  $(i + 1)^{th}$  payment (before it is paid). Micropayments may be cashed in at any time; the payer keeps track of contracts issued to guard against double spending. In this paper the fine grained payment system is implemented using the Micropayment technique. The coarse grained payment module employs a different payment technique: Here, the Bank acts as the store keeper of contract/coins with respect to client/customer and the WebCom server. The *coin* is used to credit the WebCom Server and debit the client/customer with respect to a contract. These mechanisms are described more in detail later.

## V. FINE GRAINED APPLICATION PAYMENT

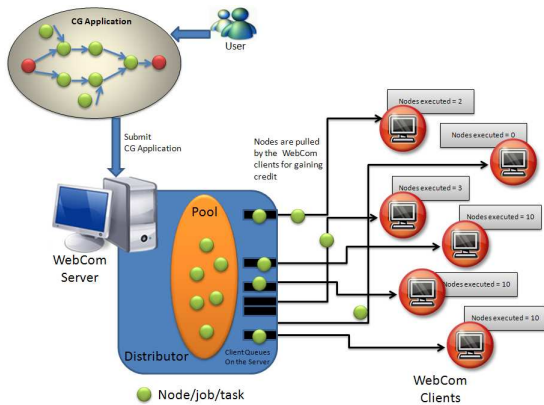


Fig. 3. Fine grained node distribution.

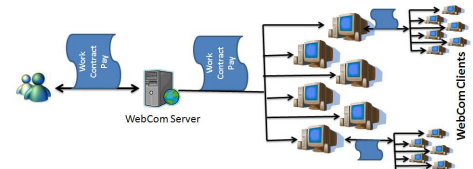


Fig. 4. Fine grained payment implementation.

### A. Fine Grained implementation

The Condensed graphs, when evaporated, can produce fine grain nodes performing tasks ranging from mathematical operations to more complex complex operations including coupling different middleware technologies and even executing complete applications.

In the fine grained implementation, Fig. 4, the complete CG application is submitted to WebCom along with the contract and the payment. There is a trust (Fig. 10) between the WebCom Server, the Bank and a Customer. This contract is analyzed by the WebCom payment module with respect to authenticity and authorization of the contract. Once the contract is accepted, WebCom unfolds the Condensed Graph

```

Authorizer: "POLICY"
Local-Constants: Bank1 = Banks public key
Conditions: (@Value * @Number <= 5000) ||
(Service == "X" || Service == "Y");
Licensees: Bank1

```

Fig. 5. Fine grained: Sample Policy stating that the bank is authorized to issue contracts

```

Authorizer: Bank1
Local-Constants: Bank1 = Bank1's public key
client = client1's public key
Conditions: @Value == 1 && @Number<=5000 && Contract == "1Ie0ohmzifz/N3JoX4m9Dw==";
Licensees: client1
Signature:Signed by the banks private key

```

Fig. 6. Sample fine grained Keynote credential

application and subcontracts each of its nodes for execution on connected clients. In this contract, the WebCom clients executing the work get the subcontracted nodes and their payment will be released when node execution is complete. There are two steps in this contract: firstly, the contract has to be accepted by the WebCom server and secondly, the WebCom Server then subcontracts the fine grained nodes to connected clients.

### B. Contract

Figure 5 describes a sample trusted Keynote credential. This credential defines the conditions under which the server allows the bank to issue contracts. These conditions are defined by using a C-like expression syntax in terms of attributes Val(value of the coin *val*), Number (number of coins *n* in the contract) and Service(Which services the server is willing to accept payments for. These services could reflect *CGs* that the server is willing to execute).

Figure 6 provides an example payment contract that a customer (public key *client1*) buys from the bank, *Bank1*. It is signed by the banks private key, delegating authority over the contract to the customer. The contract attribute provides the initial contract value  $h^n(s)$  where *s* is the secret seed known only to the customer. This particular contract is for less than or up to a maximum of 5000 coins valued at EUR1.0 each. Alternatively, the bank could decide to write a different credential that delegates to the customers the authority to generate contracts.

## VI. COARSE GRAINED APPLICATION PAYMENT

### A. Coarse Grained Implementation

In the coarse grained implementation, Fig. 8, the complete CG is submitted to WebCom along with the contract and the payment. Similar to Section V, trust is established. When the Server analyses the contract, subtasks may be scheduled to clients. In contrast, using coarse grained application payment, clients receive no payments for work they execute on the servers behalf: payment is made only to the WebCom server.

### B. Contract

Figure 9 shows a coarse grained payment credential that a customer (public key *Customer*) buys from *Bank1*. It is signed by the banks private key, delegating the contracts authority to the customer. The contract attribute provides the initial contract value  $h^n(s)$  where *s* is the secret seed known only to the

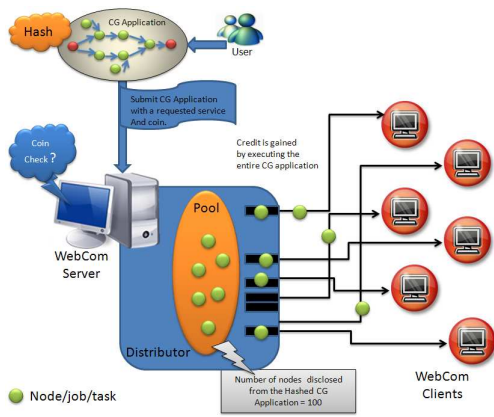


Fig. 7. Coarse grained secure application submission.

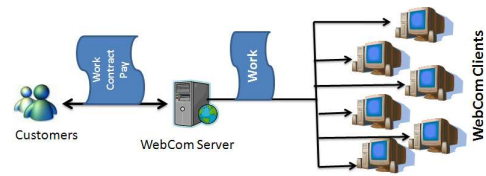


Fig. 8. Coarse grained payment implementation.

```

Authorizer: Bank1
Local-Constants: Bank1 = Bank's public key
Customer = Customer's public key
Conditions: Service == "X" && Contract == "wnrz9kYOsQl5rbNm/H9oVg==" && @Operand <= "10";
Licensees: Customer
Signature: Signed by banks private key
  
```

Fig. 9. Sample Coarse grained Keynote credential

customer, and  $n$  is the number of coins. The operand value is used to limit the work done by the client. Here, the customer has paid for the ability to execute graph  $X$  for an input value up to and including 10. This price negotiation is done between the bank and the server. The server notifies the bank of which services it will execute and their respective price range. e.g., Input value from 0 to 10, for a cost of 10, input value from 11 to 20 for a cost of 20 and so on.

## VII. WORKFLOW FOR THE FINE AND COARSE GRAINED APPROACHES

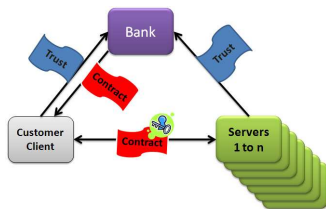


Fig. 10. Trust

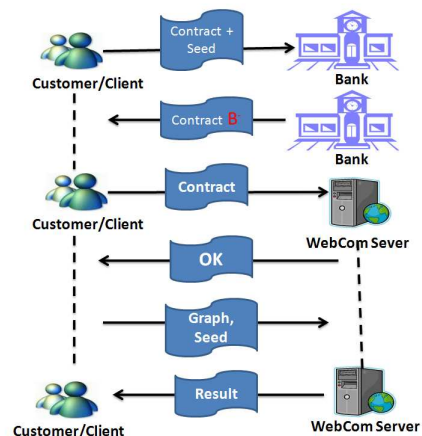


Fig. 11. Workflow.

Fig. 11 shows the flow of tasks for both the fine and coarse grained implementations. There is a trust between the bank, the customers and the WebCom resources. The flow of work through the system is as follows:

- 1) The customer sends the contract and payment to the Bank,
- 2) The Bank signs the contract and returns it to the customer,
- 3) The customer then sends the contract to the WebCom server. The WebCom server stores the contract,
- 4) The customer submits the *CG* and payment to the WebCom server,
- 5) The WebCom server executes the Graph based on the validity of the payment and sends the results back to the customer.

## VIII. CONTRASTS BETWEEN THE PAYMENT METHODS

### A. Coarse grained payment

- Jobs are not restricted by the underlying security implementations. This allows for almost no speed decrease, however, each party might either pay too much for the computation received, or get paid to little for computation given.
- There is a need to estimate the time of execution to be able to specify a price.
- There is one contract and one coin associated with this mode of payment.
- There is a Secure Socket Layer (SSL) connection between the customer and the WebCom server. Other clients connected to the WebCom server may or may not be over a secure connection.

### B. Fine grained payment

- There is an overhead in generating the coin. However, the impact of this overhead is low.
- Payment is done on a per task and not per job. This reduces the risk of doing more work than what was paid for.
- There is an SSL connection between the customer and the WebCom server, and connected clients connected.
- There is an overhead due to the SSL and Trust Management implementation.

## IX. EXPERIMENTAL SETUP AND RESULTS

The experimental testbed consisted of executing a sample application a number of times. Experimental data was collected from the execution profile of the Factorial Condensed Graph, Figure 12, when executed with parameter 1000.

The experimental configuration consisted of six Pentium IV's running at 2.6GHz with 1GB RAM. Each machine was running Fedora Linux. A two tier topology with one master and with five clients was used.

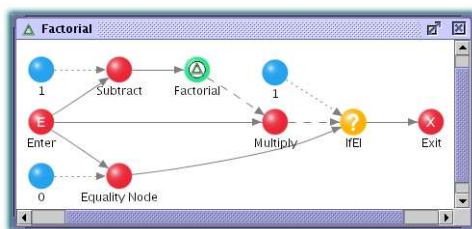


Fig. 12. factorial graph

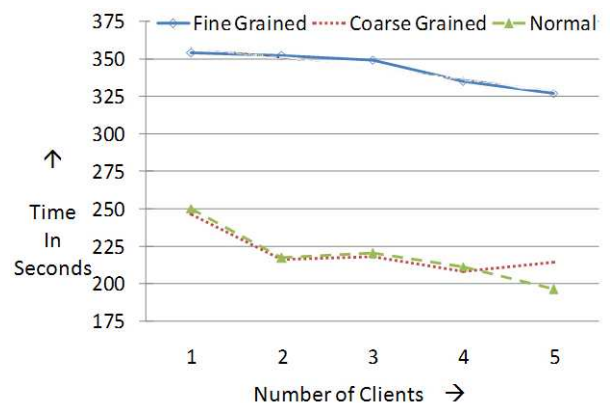


Fig. 13. Benchmarking fine,coarse and normal execution.

The object of this experiment was to ascertain the overhead in using Trust Management and SSL for fine grained execution. Due to the nature of the graph very little speedup is obtained by adding extra

machines into the computation. This is due to the fact that each iteration of the graph yields only four executable instructions. This can be clearly seen from the results shown in Figure 13.

Credentials were deployed on the clients to facilitate experimentation with both fine and coarse grained application execution.

When executing in Normal mode there is no SSL or Trust Management used between the server and clients. There is also no SSL used between the customer and server.

When executing in Coarse Grained mode, Trust Management and SSL are used only between the server and customer. There is no Trust Management and SSL between the server and clients.

Finally, when using Fine Grained mode there is full SSL and Trust management between all connections.

From the graph shown in Figure 13, it can be seen that there is little variation in the execution profile between both Normal and Course Grain modes. This is due to the fact that only one SSL and Trust Management operation is carried out when the application is executed.. The cost of this operation is negligible.

When executing in Fine Grain mode, it can be clearly seen the overhead caused by the use of SSL and Trust Managements is approximately 33%. This can also be attributed to the grain size of the task as well as the grain size of the application. Fine grained tasks in a Fine Grained application will be more costly.

Speedup was clearly affected by the type of graph selected. Although not a consideration in this example, speedup would be obtained where a graph that could better exploit parallelism would be employed. This is a subject of future work.

## X. CONCLUSION AND FUTURE DIRECTIONS

This paper briefly outlined two different payment mechanisms used for executing applications in a distributed environment. They have been implemented as fine and coarse grained application execution models. Contrasts between these two payment mechanisms were enumerated. It is expected that these methods are appropriate for Grid computing, as resources joining and utilising the Grid have varying requirements and capabilities. The payment models presented can easily cater for the differing levels of both task and resource granularity.

The experimental results show that there is a slight overhead on between the coarse payment implementation over the normal implementation (without payment). In contrast, there is a significant overhead between these and the fine grained implementation. This is primarily due to the application characteristics. It is apparent that fine grained tasks executing in a fine grained application mode (full trust management and SSL) will cost more than large grained tasks executing in the same mode. This is due to the overhead involved in carrying out Trust Management and SSL computations.

A number of areas of future work have been identified. Firstly, define a class of application that these mechanisms would be more suited towards, example highly parallel applications with large grainsize operations. Investigate the use of Subscription based computing, in particular post paid (billing) mechanisms.

## REFERENCES

- [1] Mojo nation. link: <http://sourceforge.net/projects/mojonation/>.
- [2] Public-key infrastructure (x.509) (pkix) charter. URL: [www.ietf.org/html.charters/pkix-charter.html](http://www.ietf.org/html.charters/pkix-charter.html).
- [3] N. M. A. Kupczyk and P. Wolniewicz. Simplifying administration and management processes in the polish national cluster.
- [4] A. Anandasivam and D. Neumann. Token exchange system as incentive mechanism for the e-science grid community. 2007.
- [5] R. Anderson, C. Manifavas, and C. Sutherland. Netcard – a practical electronic cash system.
- [6] C. Anglano, S.Barale, L. Gaido, A. Guarise, S.Lusso, and A. Werbrouck. An accounting system for the datagrid project- preliminary proposal draft in discussion at global grid forum 3, frascati, italy, (october 2001).
- [7] Apostolopoulos, Peris, and Saha. Transport layer security: How much does it really cost? In *INFOCOM: The Conference on Computer Communications, joint conference of the IEEE Computer and Communications Societies*, 1999.
- [8] A. Avaliani. Successful e-business systems - paypal, 2004.
- [9] A. Barmouta and R. Buyya. Gridbank: A grid accounting services architecture, 2002.
- [10] M. Blaze, J. Feigenbaum, and J. Lacy. Decentralized trust management. Technical Report 96-17, 28, 1996.
- [11] M. Blaze, J. Ioannidis, and A. D. Keromytis. Experience with the keynote trust management system: Applications and future directions.

- [12] J. P. Boly, A. Bosselaers, R. Cramer, R. Michelsen, Stig, F. Muller, T. P. Pedersen, B. Pfitzmann, P. de Rooij, B. Schoenmakers, M. Schunter, L. Vallee, and M. Waidner. The esprit project CAFE - high security digital payment systems. In *ESORICS*, pages 217–230, 1994.
- [13] R. Buyya. Economic-based distributed resource management and scheduling for grid computing, 2002.
- [14] S. N. Foley and T. B. Quillinan. Using trust management to support micropayments.
- [15] K.-M. Liew, H. Shen, S. See, W. Cai, P. Fan, and S. Horiguchi, editors. *Parallel and Distributed Computing: Applications and Technologies, 5th International Conference, PDCAT 2004, Singapore, December 8-10, 2004, Proceedings*, volume 3320 of *Lecture Notes in Computer Science*. Springer, 2004.
- [16] L. F. McGinnis, W. Thigpen, and T. J. Hacker. Accounting and accountability for distributed and grid systems. In *CCGRID '02: Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid*, page 284, Washington, DC, USA, 2002. IEEE Computer Society.
- [17] G. Medvinsky and B. C. Neuman. Netcash: A design for practical electronic currency on the internet. In *Proceedings of the First ACM Conference on Computer and Communications Security*, volume 1993, pages 102–106, 1993.
- [18] J. P. Morrison. *Condensed Graphs: Unifying Availability-Driven, Coercion-Driven and Control-Driven Computing*. PhD thesis, Technische Universiteit Eindhoven, 1996.
- [19] J. P. Morrison, B. Clayton, D. A. Power, and A. Patil. Webcom-g: Grid enabled metacomputing. *The Journal of Neural, Parallel and Scientific Computation. Special Issue on Grid Computing.*, 2004(12)(2):419–438, April 2004.
- [20] J. P. Morrison, B. C. Clayton, and A. Patil. Comparison of webcom in the context of job management systems. *Sci. Ann. Cuza Univ.*, 11:318–326, 2002.
- [21] J. P. Morrison, P. J. O’Dowd, and P. D. Healy. Searching RC5 Keyspaces with Distributed Reconfigurable Hardware. *ERSA 2003*, Las Vegas, June 23-26, 2003.
- [22] B. C. Neuman and G. Medvinsky. Requirements for network payment: The netcheque perspective. In *COMPCON*, pages 32–36, 1995.
- [23] J. S. Park and R. S. Sandhu. RBAC on the web by smart certificates. In *ACM Workshop on Role-Based Access Control*, pages 1–9, 1999.
- [24] L. Pearlman, V. Welch, I. Foster, C. Kesselman, and S. Tuecke. A community authorization service for group collaboration, 2002.
- [25] T. P. Pedersen. Electronic payments of small amounts. In *Security Protocols Workshop*, pages 59–68, 1996.
- [26] R. L. Rivest and B. Lampson. SDSI – A simple distributed security infrastructure. Presented at CRYPTO’96 Rumpsession, 1996.
- [27] M. R. Thompson, A. Essiari, and S. Mudumbai. Certificate-based authorization policy in a pki environment. *ACM Trans. Inf. Syst. Secur.*, 6(4):566–588, November 2003.
- [28] P. Xavier, W. Cai, and B.-S. Lee. Adaptive policing for token-exchange based management of shared computing resources. In *CCGRID*, pages 617–624, 2006.
- [29] J. Yu, M. Li, Y. Li, F. Hong, and Y. Du. A service-oriented accounting architecture on the grid. In *PDCAT*, pages 310–313, 2004.



**Adarsh Patil** is a Research Scholar working in one of the leading Grid and Distributed Computing research group, Centre for Unified Computing, Cork, Ireland. His areas of interests include Grid & Distributed Computing and Economic Models in Grid Environment. He has a Bachelor’s degree in Computer Science and Engineering from University B.D.T College of Engineering (Kuvempu University), Davangere, Karnataka State, India. He is currently pursuing his PhD in Department of Computer Science, University College Cork, Cork, Ireland. He has reviewed for a number of journal publications and has been a program committee member, session chair and organizing committee member for a number of conferences. Mr. Patil has one of his articles published in *Grid Technologies Emerging from Distributed Architectures to Virtual Organizations*, Vol. WIT Press 2006, PP. 271-307, Editors: M.P. Bekakos, G.A. Gravvanis and H.R. Arabnia under the chapter name WebCom-G: Middleware to Hide the Grid.



**Christoffer Norvik** is a Research Scholar working for the Grid and Distributed Computing research group, Centre for Unified Computing, Cork, Ireland. His areas of interests include Grid & Distributed Computing and Security. He has a Bachelor’s degree in Computer Science from University College Cork, Cork, Ireland. He is currently pursuing his MSc in the Department of Computer Science, University College Cork, Cork, Ireland.



**David A. Power** is a Post Doctoral research fellow at the Centre for Unified Computing in University College Cork, Ireland. David obtained his PhD. in 2004. He co-developed the WebCom system and is now working on the WebCom-G project. His main areas of interest are scheduling techniques on distributed systems and creating distributed workflows. He has reviewed for a number of journal publications and has been a program committee member, session chair and organizing committee member for a number of conferences. He has also been a lecturer in the Computer Science Department where he prepared and delivered courses in programming and Software Engineering. David is a member of the IEEE, IEEE Computer Society and the ACM.



**John P. Morrison** is the founder and director of the Centre for Unified Computing. He is a cofounder and co-director of the Boole Centre for Research in Informatics and a cofounder and co-director of Grid-Ireland. Prof. Morrison is a Science Foundation of Ireland Investigator award holder and has published widely in the field of Parallel Distributed and Grid Computing. He has been the guest editor on many journals including the Journal of Super Computing and the Journal of Scientific Computing. He is on the Editorial Board of Multi-Agent and Grid Systems: An International Journal, published by ISO Press, and the International Journal of Computational Intelligence: Theory and Practice (IJCITP). He is a member of the ACM and a senior member of the IEEE. Prof Morrison is a member of the I2Lab Advisory Board in the University of Central Florida. He has served on dozens of international conference programme committees and is a co-founder of the International Symposium on Parallel and Distributed Computing.