A Multi-Commodity Flow Approach to Maximising Utility in Linked Market-Based Grids

James Broberg
Grid Computing and Distributed Systems (GRIDS) Laboratory
Department of Computer Science and Software Engineering
The University of Melbourne, Australia
brobergj@csse.unimelb.edu.au

Rajkumar Buyya
Grid Computing and Distributed Systems (GRIDS) Laboratory
Department of Computer Science and Software Engineering
The University of Melbourne, Australia
raj@csse.unimelb.edu.au

ABSTRACT

In this paper we consider the problem of maximising utility in linked market-driven distributed and Grid systems. In such systems, users submit jobs through brokers who can virtualise and make available the resources of multiple service providers, achieving greater economies of scale, improving throughput and potentially reducing cost. Customers compete against each other by assigning a utility value or function to the successful processing of their jobs in an effort to have them prioritised in the face of contested and constrained resources. Brokers and service providers also attempt to maximise the utility they gain, choosing to process jobs that will earn them the highest profit with respect to the resources required. For this to be effective over many linked computing marketplaces highly distributed resource allocation is needed, where each participant can operate independently using only local information, and ideally reach a global state where all participants are satisfied. We model such a system by adapting the classical multi-commodity flow problem to the market-based, utility driven distributed systems, where all participants selfishly attempt to maximise their own gain. We then obtain a utility-aware distributed algorithm that generates increased utility for participants in such systems, especially under scenarios of high contention.

Categories and Subject Descriptors

C.2.4 [**COMPUTER-COMMUNICATION NETWORKS**]: Distributed Systems—*Distributed applications*

1. INTRODUCTION

We are motivated by the problem of maximising *utility* for participants in linked market-driven distributed [11, 9, 15] and Grid [5] systems. In such systems (shown in Figure 1), users have jobs that need to be processed, for which they are willing to proportionally compensate a provider to perform depending on the utility they receive.

Rather than dealing with service providers directly, users facilitate access through brokers (like the Gridbus Broker [13]) who can virtualise and make available the resources of multiple heterogeneous service providers, achieving greater economies of scale, improving throughput and reducing complexity for users.

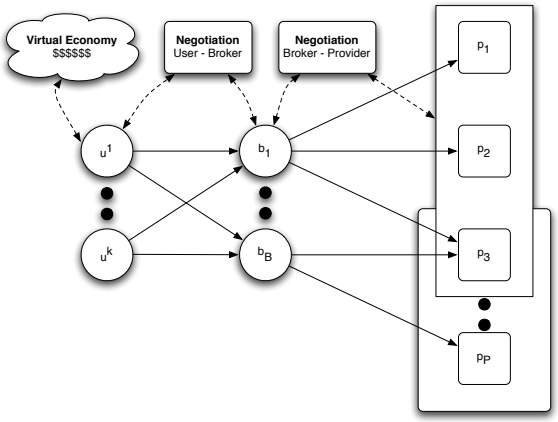

Figure 1: A Market-Based Distributed System

Despite the advantages of market-driven systems, there is considerable complexity involved in ensuring each participant is satisfied. Clients need to negotiate with brokers to obtain the resources they need to process their jobs. They do this by setting a fixed value or utility function describing what they are willing to pay for one or many resources needed to process their job. The broker must carefully choose between bids from many clients with the aim of maximising the revenue it receives. However, finding the optimum selection of users to service is an NP-hard problem. The broker also continuously negotiates with different service providers to lease a set of resources to provide the actual computation for user's jobs. The broker pays the service providers for access to their resources, whilst aiming to make a reasonable profit from the value-added reselling of them to users. Brokers and service providers attempt to maximise the utility they gain, by choosing to accept and process the jobs that will earn them the highest revenue or yield with respect to the resources required.

With the emergence of numerous commercial [1, 9] and experimental [2, 6] computing marketplaces that can potentially be cherry-picked from and aggregated together, it is clear that a co-operative, centralised solution for assigning resources is not appropriate, let alone tractable in this environment. We cannot mandate participants in such systems to behave in a certain manner to maximise

MGC '07, November 26, 2007 Newport Beach - CA, USA

the utility of the system as a whole. Instead, we expect each participant to behave in a selfish, autonomous manner, where their goal is maximising their own gain [7]. Highly distributed solutions are therefore required in order for users, brokers and providers to maximise their benefit from participating in such systems. The logic of such solutions needs to be trivially embeddable inside user agents, broker middleware and service provider resource management systems.

We propose to model these systems as an adaptation of the classical multi-commodity flow problem [4] to the market-based, utility driven problem. In the traditional multi-commodity flow problem, there are several "commodities", which have a source (origin) and a sink (destination). They each compete for capacity over bandwidth-constrained edges in a network (represented by a graph) when routing flow from the source to the sink. Awerbuch and Leighton devised a distributed potential energy argument to move flow in a multi-commodity flow system, allowing each node to operate in a distributed manner whilst still performing within some small $(1+\epsilon)$ factor of the global optimal solution [3]. This high-level approach has been successfully adapted to a variety of problem domains, including maximising lifetime routing in wireless ad-hoc and sensor networking [12], and enabling distributed resource allocation (both bandwidth and CPU) in stream processing systems [14]. In this paper we explore the use of a similarly distributed approach in performing resource allocation, to maximise each participants utility in linked market-based distributed and grid systems.

2. RELATED WORK

Resource allocation and scheduling algorithms are increasingly incorporating market inspired techniques as an alternative to traditional approaches. These market-based techniques look beyond the typical metrics that motivate system designers which focus on maximising throughput, and minimising response time and slowdown. Rather, they focus on a user's *utility*, which can be represented by the value they place on successful processing of their job.

One such market-based technique is Bellagio [2] - a distributed resource discovery and market-based allocation system. Users identify resources of interest via a resource discovery mechanism, and register their preference (via virtual currency) for said resources over time and space using combinatorial auction bids. Bellagio assumes that an authentication entity exists that authenticates bids, resource capabilities (reservations) and account balances. The Bellagio system uses a "second-price" style auction to reveal users true value for goods. The system controls the distribution of wealth in the virtual economy (e.g. when to inject currency into system and how much), which subsequently controls the share of resources received by participating users.

Mirage [6] is very similar system to that of Bellagio - however it is a real-world deployed system, which revealed many users behaving strategically to exploit it. Other novel features (over Bellagio) include a proportional share profit sharing, where proceeds from cleared auctions are distributed proportionally to idle users to accumulate ad-hoc credit, and a savings tax to address unbalanced usage patterns, where credit regresses back to a baseline over time. The authors found that the values that users placed on resources varied over four orders of magnitude, validating the market-based approach. Such differences in user valuations cannot be captured by traditional utility oblivious scheduling and resource allocation techniques.

There are a number of restrictions in the above systems. These systems are isolated markets, controlled by a single entity. It is unclear how these allocation techniques would operate within multiple linked marketplaces. The algorithms themselves are highly centralised, with authentication, auction clearing and resource reservation all handled by a central body. Clearing combinatorial auction bids is an NP-hard problem, which is not appropriate for systems with even a moderate amount of bids. As such, sub-optimal approximation algorithms must be used to clear these auctions.

Many utility aware scheduling and admission control algorithms have been proposed in recent years, such FirstPrice/FirstReward [8], and FirstProfit/FirstOpportunity [11]. However, these approaches tend to be user centric with less consideration to the utility and behaviour of the brokers and service providers in market-driven distributed systems. In these approaches, processing delays incurs an immediate cost to provider. Tasks are also assumed to be preemptible and resumed with negligible or no cost. However, this is not valid for many data and memory intensive applications. These algorithms generally model only one broker and market at a time, where optimal schedules and admission control are computed centrally rather than in a distributed manner.

In the last 3 years there has been an increased research focus on the notion of applying Austrian economist F.A. von Hayek's notion of a 'Catallaxy' and applying it to market-driven grid computing. The idea of 'Catallactics' considers markets where prices evolve from the actions of economically self-interested participants. Each participant tries to maximise their own gain whilst having limited information available to them.

Eymann et al. have investigated the issues and requirements of implementing an electronic grid market based on the concept of 'Catallaxy' [7], a 'free market' economic self-organisation approach. However, they found that solving this problem is a complex multi-attribute allocation problem, found to be NP-complete in previous work by other researchers in the field. The authors note that in interrelated markets, the allocation of resources and services in one market invariably influences the outcomes in the other market. These interactions should occur without relying on a traditional single centralised broker. Instead, participants should be self-organising and follow their own interest, maximising their own utility. A catallaxy approach works on the principle that there are autonomous decentralised agents, which have constant negotiation and price signalling occurring between them. Indeed, we concur that changing conditions (availability, competition) on the resource market should be reflected by cascading 'pressure' (via price changes or other signals) that reflect the respective scarcity and demand for a resource. Participants need to constantly read these signals and react accordingly to maximise their own utility.

3. MARKET-BASED DISTRIBUTED COMPUTING MODEL

We choose to represent the market-based distributed computing system as a graph $\mathcal{G} = (\mathcal{N}, \mathcal{E})$, depicted in Figure 2. The set of $\mathcal{N}$ nodes consists of source nodes, intermediary nodes and sink nodes. The set of $\mathcal{E}$ edges represent the links over which job streams are transferred for eventual processing at the service providers. In our problem domain, the source nodes map to users that submits a job stream to the system, denoted by the subset of nodes $\mathcal{N}_s$. The intermediary nodes consist of brokers and service providers, with the subsets denoted by $\mathcal{N}_b$ and $\mathcal{N}_p$ respectively. The broker can forward jobs to one or many service providers that it is connected to. The service provider nodes process job streams received from the brokers, and pass the resulting output to sink nodes (denoted by $\mathcal{N}_t$) that act as a collection point. We note that $\mathcal{N} = \mathcal{N}_s \cup \mathcal{N}_b \cup \mathcal{N}_p \cup \mathcal{N}_t$. Each node $n \in \mathcal{N}$ has constrained computational resources R_n. Each pair of connected nodes (u, v) joined by edge $(u, v) \in \mathcal{E}$ has constrained bandwidth $B_{u,v}$.

The job streams generated by each client are denoted as unique *commodities*. There are K commodities, each originating from a unique source node s^k and collated at a unique sink node t^k, where $k = 1, ..., K$. These job streams arrive at a rate of d^k per time unit, and have a service requirement of r^k per job unit. We can model both data intensive and compute intensive workloads by simply varying the value of r^k, with a high value ($>> 1$) representing a compute intensive job stream and a low value ($<< 1$) representing a data intensive job stream.

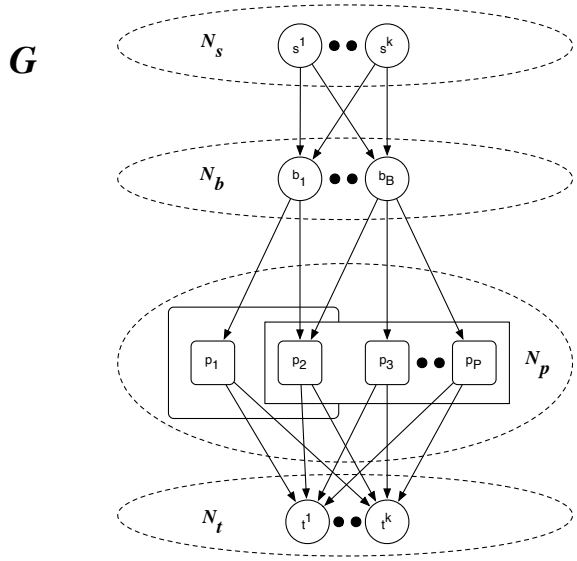

Figure 2: Graph Representation

4. STATIC PROBLEM FORMATION

4.1 Node Responsibilities and Behaviour

Each node in the system acts in a self-interested manner, and seeks to selfishly maximise its own utility at all times subject to the limited computational and bandwidth resources it has available to it.

4.1.1 User Nodes

There are K user nodes in the system, with each user node $s^k \in \mathcal{N}_s$ generating a unique job stream (or *commodity*) of type k, where $k = 1, 2, ..., K$. The node s^k has no predecessors in the graph $\mathcal{G}$, and we denote $\mathcal{O}(s^k)$ as the set of all successor (downstream) nodes of node s^k. These user nodes are analogous to source nodes in traditional multi-commodity flow systems. We denote $x^k_{s,b}$ as the number of jobs submitted to downstream broker node $b \in \mathcal{O}(s^k)$ from user node s^k. A user k has a private valuation v^k_s that they place on each successfully processed job unit by a broker. This is, in effect, the *true value* they place on that particular job stream. A specific broker node b charges the user a value of v^k_b per job unit to co-ordinate the actual processing. A user will attempt to maximise their own utility U^k_s, where $U^k_s = v^k_s - v^k_b$, by directing jobs to brokers that minimise the cost where possible.

4.1.2 Broker Nodes

There are B broker nodes in the system, with each broker node b (where $b = 1, 2, ..., B$) receiving job streams from its connected predecessor (upstream) customers, denoted by $\mathcal{I}(b)$. The broker node b directs these jobs to the set of successor (downstream) service providers for which it has a relationship with (represented by a connected edge in the graph $\mathcal{G}$), and denoted as $\mathcal{O}(b)$. The number of jobs processed at broker b for submission to downstream processing node p for user k is denoted as $x^k_{b,p}$, where $p \in \mathcal{O}(b)$. These nodes perform minimal processing on the incoming job stream, rather they simply direct them to provider nodes. As such, the processing requirements are modelled as $r_{b,p} = 1$ for all jobs passing through, representing the nominal effort required to direct job streams to downstream providers. A broker b has finite bandwidth on the edges connecting to upstream nodes $\mathcal{I}(b)$ and downstream nodes $\mathcal{O}(b)$. Each successfully provisioned job unit for user node s^k generates a revenue of v^k_b for the broker, paid by the user node. As the broker must pay the service provider $p \in \mathcal{N}_p$ an amount of v^k_p per job unit to perform the actual processing, the utility (i.e. profit) generated for the broker is $U^k_b = v^k_b - v^k_p$.

4.1.3 Service Provider Nodes

There are P service provider nodes, where each service provider node p (where $p = 1, 2, ..., P$) processes jobs assigned to them from predecessor (upstream) broker nodes, denoted as $\mathcal{I}(p)$. The number of jobs processed at service provider p for collection at downstream collection node c for user k is denoted as $x^k_{p,c}$, where $c \in \mathcal{O}(p)$. Each job unit for user k at node p requires $r^k_{p,c}$ resources to process for downstream collection node $c \in \mathcal{O}(p)$. Processing on service provider nodes is constrained, with each node p having R_p processing resources available. Each successfully processed job unit for user k generates a value of v^k_p for the service provider, which is paid by the broker. Therefore, the utility generated by the broker is $U^k_p = v^k_p$.

4.1.4 Collection Nodes

The collection node t^k collects flow for each commodity $k \in K$ from the upstream service providers $\mathcal{I}(t^k)$. As no actual processing occurs here, we model this as a node with unlimited processing resources, where $R_c = \infty$. These collection nodes are analogous to sink nodes in traditional multi-commodity flow systems.

4.1.5 Bandwidth Nodes

To model bandwidth costs, we utilise an extended graph representation as described in prior work on distributed multi-commodity flow algorithms adapted to stream processing systems [14]. That is, for any two connected nodes u and v, a bandwidth node $n_{u,v}$ is introduced to model the transfer of flows. Given $(u, v) \in \mathcal{E}$ with bandwidth $B_{u,v}$, in the new expanded graph $\mathcal{G}' = (\mathcal{N}', \mathcal{E}')$ this bandwidth constrained edge is now represented by directed edges $(u, n_{u,v})$ and $(n_{u,v}, v)$, and bandwidth node $n_{u,v}$. Bandwidth node $n_{u,v}$ has constrained resources $R_{u,v} = B_{u,v}$. This technique unifies the problem at each node from two resource constraints (compute and bandwidth) to a single generic resource constraint problem. Each node, be it a user, broker, provider, collection or bandwidth node, processes jobs that maximise its utility subject to its specific role-based resource constraint. We will refer to the expanded graph $\mathcal{G}' = (\mathcal{N}', \mathcal{E}')$ exclusively from now on.

4.2 Linear Program Formation

We now formulate the problem as a static linear program. Using the expanded graph representation described previously, we denote $\mathcal{I}'(n)$ as the set of all predecessor nodes and $\mathcal{O}'(n)$ as the set of all successor nodes of a given node $n \in G'$. We denote $x^k_{n,n'}$ as the amount of job units from user k to be processed at node $n \in N'$, generating output to downstream node $n' \in \mathcal{O}'(n)$.

A feasible solution must satisfy these conditions:

$$x^k_{n,n'} \geq 0, \quad \forall k; n, n' \in \mathcal{N}' \tag{1}$$

$$\sum_k \sum_{n' \in \mathcal{O}'(n)} x^k_{n,n'} r^k_{n,n'} \leq R_n, \quad \forall n \in \mathcal{N}' \tag{2}$$

$$\sum_{n' \in \mathcal{O}'(n)} x^k_{n,n'} - \sum_{n'' \in \mathcal{I}'(n)} x^k_{n'',n} = \begin{cases} f^k, \text{if } n = s^k \\ -f^k, \text{if } n = t^k \\ 0, \text{otherwise} \end{cases} \quad \forall k; n \in \mathcal{N}' \tag{3}$$

Condition (1) specifies that the flow of job units must be non-negative. Given that the graph is directed, all flows must move downstream. Condition (2) ensures that the resource constraints at each node (regardless of it's role) are respected. Condition (3) ensures that flow balance occurs, such that jobs enter each node at the same rate they leave for each user.

4.2.1 Maximum Feasibility Problem

As a baseline, we first consider the basic feasibility problem, where each user k has a demand d^k, $k = 1, 2, ..., K$.

$$f^k = \delta d^k, \quad \forall k; \quad \text{and (1)-(3).}$$

In this scenario, we simply want to find the maximum fraction δ of each users demand d^k to be simultaneously satisfied without consideration of the utility gained by any participant in the system.

4.2.2 Maximum Utility Feasibility Problem

In the maximum utility feasible flow problem, we can find the allocations that will maximise the *overall* utility in the system, whilst still satisfying all feasible demands.

$$\max \sum_{k=1}^{K} \sum_{n' \in \mathcal{O}'(n)} U^k_{n,n'} x^k_{n,n'} \quad \text{s.t. } f^k = \delta d^k, \quad \forall k; \quad \text{and (1)-(3).}$$

where δd^k is the maximum feasible input rate of job units by user k found in the feasibility problem. This represents the maximum utility that could be generated by a system if all nodes co-operated, but is unlikely to be achieved given all nodes act in a self-interested manner, maximising their own individual rather than global utility.

5. DISTRIBUTED PROBLEM FORMATION

We assign both output and input queues for each user k's jobs at the tail and head of each edge $e \in \mathcal{E}'$. We denote the corresponding queue heights as $q^k(e^t)$ and $q^k(e^h)$ respectively. These queue heights are normalised by demand, where $\bar{q}^k(e^h) = q^k(e^h)/d^k$ and $\bar{q}^k(e^t) = q^k(e^t)/d^k$ are the relative heights. This avoids unfairness in the system that could be caused by one user flooding the system with jobs and dominating the resources.

We define a potential function $\Phi(\bar{q})$ associated with each queue, where the potential function is twice-differentiable and convex. For analytical convenience, in this paper we utilise the potential function $\Phi(y) = \frac{y^2}{2}$. More aggressive potential functions could be used such as the exponential function, that could allow faster convergence, but they complicate the problem formation. We define $\Phi^k(e) = \Phi(\bar{q}^k(e^t)) + \Phi(\bar{q}^k(e^h))$, for any edge e and commodity k. The potential of a given node $n \in \mathcal{N}'$ is defined as $\Phi(n) = \sum_{k=1}^{K} \sum_{n' \in \mathcal{O}'(n)} \Phi^k(e_{n,n'})$, where $e_{n,n'}$ denotes edge (n, n'). The potential of the entire system, Φ, is simply $\Phi = \sum_{n \in \mathcal{N}'} \Phi(n)$. The algorithm will then decide locally the optimal $x^k_{n,n'}$ values, representing the amount of job units from user k to move across edge (n, n'), with the aim to minimise the potential of the entire system and maximise the utility-weighted flow processed.

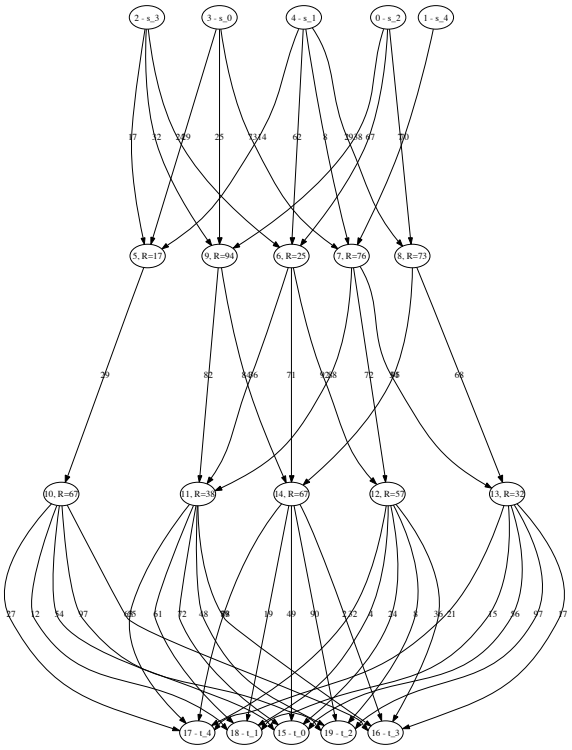

Figure 3: An example scenario with 5 users, 5 brokers and 5 service providers

The algorithm is continuously executed in a sequence of rounds performed at each node simultaneously and independently. In each round, four phases are performed as described in the next section.

5.1 Distributed Solution

1. For each commodity k, inject d^k job units at its corresponding source s^k.

2. At every node $n \in \mathcal{N}'$, balance the queues for each commodity ($\forall k$) to equal heights.

3. For every node $n \in \mathcal{N}'$, push $x^k_{n,n'} \geq 0$ amount of commodity k job units across edge (n, n') for all $n' \in \mathcal{O}'(n)$, (let $e_{n,n'}$ denote the edge (n, n')), so that

$$\min \sum_k \sum_{n' \in \mathcal{O}'(n)} U^k_{n,n'} \Big[\Phi(\bar{q}^k(e^t_{n,n'}) - \bar{x}^k_{n,n'}) + \Phi(\bar{q}^k(e^h_{n,n'}) + \bar{x}^k_{n,n'}) \Big] \tag{4}$$

$$\text{s.t.} \sum_k \sum_{n' \in \mathcal{O}'(n)} d^k \cdot \bar{x}^k_{n,n'} \cdot r^k_{n,n'} \leq R_n, \tag{5}$$

where $\bar{x}^k_{n,n'} = x^k_{n,n'}/d^k$. If $n' = t^k$, we set the second term in (4) to be zero as the potential at sink node t^k is always zero.

4. Absorb job units that have reached its collection point.

5.2 Per-node Utility Optimal Resource Allocation

The most important stage of the distributed solution is Phase 3, where the local optimisation problem defined by (4) and (5) is solved at each node n. As such, every node n will allocate its resources so as to minimise the utility weighted potential at node n subject to the resource constraint R_n. Using Lagrangian multipliers, the optimal solution must satisfy

$$\bar{x}^k_{n,n'} = \max\{\frac{\Delta^k(e_{n,n'}) - \frac{sd^k \cdot r^k_{n,n'}}{U^k_{n,n'}}}{2}, 0\},$$

where $\Delta^k(e_{n,n'}) = \bar{q}^k(e^t_{n,n'}) - \bar{q}^k(e^h_{n,n'})$, and $s(\geq 0)$ is the Lagrangian multiplier. The optimal value of s is the minimum $s \geq 0$ such that (5) is satisfied. That is,

$$\sum_k \sum_{n' \in \mathcal{O}'(n)} d^k r^k_{n,n'} \max\{\frac{\Delta^k(e_{n,n'}) - \frac{sd^k \cdot r^k_{n,n'}}{U^k_{n,n'}}}{2}, 0\} \leq R_n.$$

The solution can then be obtained by modifying the reverse 'water-filling' method described in previous work [14] to consider the locally optimal resource allocation that maximises a participants utility, where $h^k_{n,n'} = \frac{U^k_{n,n'}\Delta^k(e_{n,n'})}{d^k r^k_{n,n'}}$, and $a^k_{n,n'} = \frac{(d^k r^k_{n,n'})^2}{2U^k_{n,n'}}$.

6. RESULTS

In this section we present a subset of the results we have observed from our network flow simulator, implementing the algorithm described in Section 5. Our simulator allows us to generate arbitrary random graphs with K users, B brokers and P service providers. We set the probability of connectivity between users and brokers, and brokers and service providers to equal $\frac{\log n}{n}$. Demands d^k and resource requirements r^k are uniformly generated, and user valuations are generated from a highly variable Bounded Pareto distribution, $BP(k, p, \alpha)$, to reflect the variation of valuations that participants place on jobs, that is not necessarily correlated with the service and bandwidth requirements [6]. We allow job valuations to vary over three orders of magnitude, from 1 to 1000, with an α parameter of 1 to reflect this. The available resources at each node n and the bandwidth on the connecting edges (u, v), denoted as R^n and $B_{u,v}$ respectively, are drawn from a uniform distribution.

For each scenario generated, the Feasibility Problem described in Section 4.2.1 is solved using a Linear Program solver [10], obtaining the maximum concurrent demands (δd^k) that can be feasibly met by the system. The distributed solution described in Section 5.1 is then executed, for both the Feasibility Problem and the Maximum Utility Feasibility problem. In the case of the former, for the purpose of calculating flows at each node (described in Phase 3. of the distributed algorithm), all utilities ($\forall \sum_{k=1}^{K} \sum_{n' \in \mathcal{O}'(n)} U^k_{n,n'}$) are equal to 1. For the latter, the actual utilities are utilised to calculate flows in a utility-effective manner. In both cases, utility is generated at the same rates for flows moving over the same edge, for a given user's job stream. In both algorithms, all feasible demands d^k are met for each user k.

In Figure 3 we show one example scenario with 5 users, 5 brokers and 5 service providers. Figures 4 and 5 show the execution of the standard feasible algorithm and the utility-aware feasible algorithm respectively. In both cases we can see that for each demand d^k (represented by the horizontal lines), a feasible flow $f^k = d^k$ has been found. However, despite both algorithms finding a feasible flow, the utility aware algorithm generated approximately 20% more utility overall, highlighted in Figure 6.

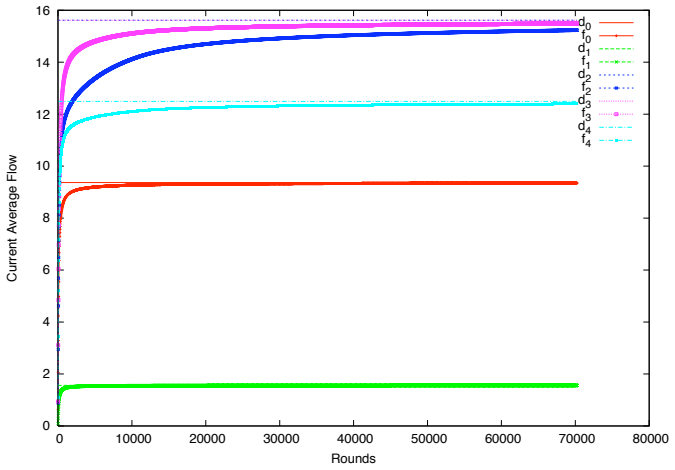

Figure 4: Feasible (fair) resource allocation

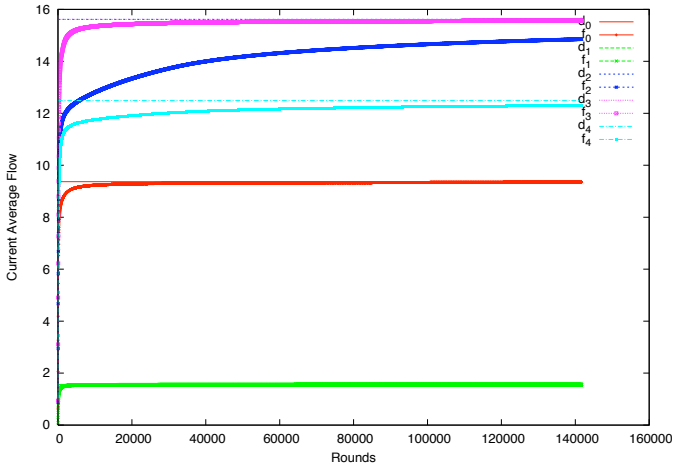

Figure 5: Utility-driven Feasible resource allocation

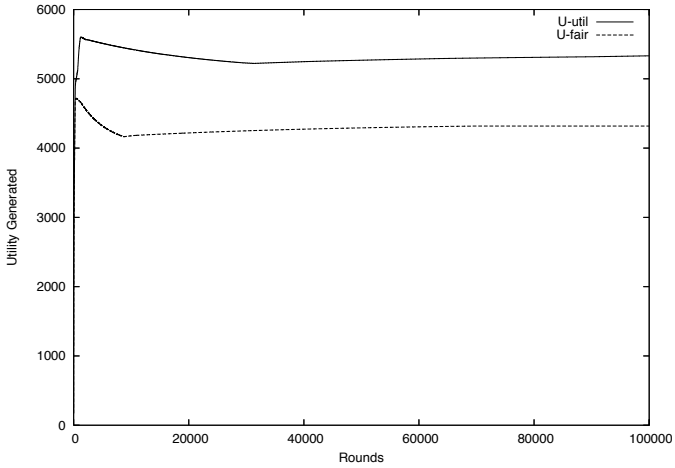

Figure 6: Comparison of utility generated in the system

Table 1: Utility increase

Users	Brokers	Providers	Utility Gained	Global Utility
5	5	5	16.2%	62%
10	5	5	18.7%	60%
20	5	5	42.5%	61%

We now consider the effect that increased competition for resources has on the utility generated in the type of market-based systems we are concerned with in this paper. Whilst keeping the number of brokers and service providers fixed (at 5 each), we increase the number of users competing for resources. We explore the scenarios where 5, 10 and 20 users are competing for resources, and examine the increase in utility gained by our utility-aware algorithm compared to a standard, 'fair' flow balancing algorithm, as well as the percentage of global utility (defined in 4.2.2) achieved by our selfish participants. In each instance, we generate 100 random problem scenarios, and look at the average utility gain for each scenario. In both scenarios, all feasible demands d^k *must* be met in all problem scenarios generated, and the algorithms execute until this is the case. Despite increases in competition, in all instances there exists feasible paths to satisfy demands.

A summary of these results is presented in Table 1. In the case of 5 user nodes, there is only minimal contention at the brokers and service providers, allowing user's job streams to find paths through the system relatively easily. Due to the nature of flow balancing algorithms, the utilities only have a minor impact in the absence of significant competition and contention at broker and provider nodes. Despite this, a small overall improvement of 16.2% is observed. As the number of user nodes increases to 10 there is a further increase in the utility generated by the utility-aware algorithm over the standard feasible (fair) flow algorithm. We double the number of nodes again, to consider a scenario of 20 user nodes. In this case we observe a more significant increase in utility, as the utility functions of users, brokers and service providers are increasingly utilised to allocate resources in the face of constrained resources. As such, all participants will favour job streams that increase their own utility, forcing low utility streams to take an alternate path through the system. An overall utility gain of 42.5% is observed in this scenario. In all scenarios, the percentage of global utility satisfied remains constant at around 60%.

7. CONCLUSION & FUTURE WORK

In this paper we considered the problem of maximising utility in linked market-driven distributed and Grid systems. We modelled this system as an adaptation of the classical multi-commodity flow problem to the market-based, utility driven distributed systems. For systems under high contention, we observed significant increases in utility gained by utility-aware resource allocation, where finite resources are allocated with consideration to the utility generated for processing specific job streams. However, we note that the percentage of global (optimal) utility could be improved significantly. Under low contention, the utility functions have a lower impact on resource allocation decisions, with the resulting utility generated tending toward that of a standard feasible flow balance algorithms. Examining the appropriateness of adapting min-cost multi-commodity flow algorithms (i.e. max-cost, 'max-utility') is a priority, to ensure maximum utility paths are utilised even in the absence of any competition. We also intend to examine the effect of a managed virtual economy on the behaviour of utility-aware flow algorithms, with users only having a fixed pool of currency to spend in order to increase the utility associated with their job, with currency allocated to users at fixed or variable intervals.

8. ACKNOWLEDGEMENTS

This work is supported under the Australian DEST funded International Science Linkage project, "Utility Grid".

9. REFERENCES

[1] Amazon.com, Inc. Amazon Elastic Compute Cloud (Amazon EC2), 2007. Available at http://aws.amazon.com/ec2.

[2] A. AuYoung, B. Chun, A. Snoeren, and A. Vahdat. Resource allocation in federated distributed computing infrastructures. In *Proc. of the 1st Workshop on Operating System and Architectural Support for the Ondemand IT InfraStructure*, October 2004.

[3] B. Awerbuch and T. Leighton. A simple local-control approximation algorithm for multicommodity flow. In *Proc. of 34th Annual Symposium on Foundations of Computer Science*, pages 459–468, November 1993.

[4] M. S. Bazaraa, J. J. Jarvis, and H. D. Sherali. *Linear programming and network flows (3rd ed.)*. John Wiley & Sons, Inc., New York, NY, USA, 2005.

[5] R. Buyya, D. Abramson, and S. Venugopal. The grid economy. *Proc. of the IEEE*, 93(3):698– 714, March 2005.

[6] B. N. Chun, P. Buonadonna, A. AuYoung, C. Ng, D. Parkes, J. Shneidman, A. Snoeren, and A. Vahdat. Mirage: A microeconomic resource allocation system for sensornet testbeds. In *EMNETS 2005: Proc. of the 2nd IEEE Workshop on Embedded Networked Sensors*, May 2005.

[7] T. Eymann et al. Catallaxy-based grid markets. *Multiagent Grid Systems*, 1(4):297–307, 2005.

[8] D. E. Irwin, L. E. Grit, and J. S. Chase. Balancing risk and reward in a market-based task service. In *HPDC '04: Proc. of the Int. Symp. on High Performance Distributed Computing*, pages 160–169, Washington, DC, USA, 2004.

[9] K. Lai, L. Rasmusson, E. Adar, L. Zhang, and B. A. Huberman. Tycoon: An implementation of a distributed, market-based resource allocation system. *Multiagent Grid Systems*, 1(3):169–182, 2005.

[10] R. Lougee-Heimer. The common optimization interface for operations research. *IBM Journal of Research and Development*, 47(1):57–66, January 2003.

[11] F. I. Popovici and J. Wilkes. Profitable services in an uncertain world. In *SC '05: Proc. of the 2005 ACM/IEEE conf. on Supercomputing*, page 36, Washington, DC, USA, 2005. IEEE Computer Society.

[12] A. Sankar and Z. Liu. Maximum lifetime routing in wireless ad-hoc networks. In *INFOCOM 2004: Proc. of Twenty-third Annual Joint conf. of the IEEE Computer and Communications Societies*, volume 2, pages 1089–1097, March 2004.

[13] S. Venugopal, R. Buyya, and L. Winton. A grid service broker for scheduling distributed data-oriented applications on global grids. In *MGC '04: Proc. of the 2nd workshop on Middleware for grid computing*, pages 75–80, New York, NY, USA, 2004. ACM Press.

[14] C. H. Xia, J. A. Broberg, Z. Liu, and L. Zhang. Distributed resource allocation in stream processing systems. In *DISC 2006: Proc. of 20th Int. Symposium on Distributed Computing (LNCS 4167)*, pages 489–504, 2006.

[15] C. S. Yeo and R. Buyya. A taxonomy of market-based resource management systems for utility-driven cluster computing. *Software Practice and Experience*, 36(13):1381–1419, 2006.