

# Resource Allocation in Streaming Environments

Lu Tian and K. Mani Chandy

Computer Science 256-80

California Institute of Technology

Pasadena, California 91125 USA

lutian@cs.caltech.edu

mani@cs.caltech.edu

**Abstract**—This paper considers resource allocation algorithms for processing streams of events on computational grids. For example, financial trading applications are executed on large computational grids that receive streams of data such as stock ticker prices, commodity prices, foreign-exchange rates and total risk exposure. The economic value of a computation depends on the time taken to execute it; an arbitrage opportunity can disappear in seconds. Given limited resources, it is not possible to process all streams without delay. The more resource available to a computation, the less time it takes to process the input, and thus the more value it generates. Therefore, the scheduling policy should be designed to optimize the net economic value of computations executed on the grid. In this paper, we propose two scheduling/resource allocation algorithms for processing streams on computational grids to optimize economic value. Both algorithms are based on market mechanisms; one uses a centralized market and the other decentralized markets. We prove bounds on performance and present measurements to show that the performances of the resource allocation systems are near-optimal and outperform load-balancing heuristics.

## I. INTRODUCTION

In the last decade, the proliferation of the Internet, the World Wide Web, and sensor networks has fueled the development of applications that process, analyze, and react to continuous data streams in a near-real-time manner. Examples of such stream applications include network traffic monitoring, intrusion detection, financial services, large-scale reconnaissance, and surveillance. Computational grids form the critical infrastructure for many businesses, just as automobile factories did for Ford and chemical plants for Du Pont; therefore, designing grids in a way that maximizes profits for economic enterprises is important. Current state-of-the-art management systems for computational grids include Legion [1] and Globus [2]. The substantial existing research on grid management can be applied to many economic enterprises by treating the grid as the physical infrastructure that generates economic value. The allocation of resources in a computational grid to generate the most economic value from streaming applications presents an interesting research problem.

The resource allocation problem is one of the oldest and most thoroughly studied problems in computer science. It is proven to be NP-complete and computationally intractable [3], [4]. Thus, any practical scheduling algorithm presents a trade-off between its computational complexity and its performance [5], [6]. There have been several good comparisons of commonly used algorithms [7], [8], [9]. The most common formulation of the problem in a distributed computing environ-

ment is: given a set of tasks, each associated with a priority number and a deadline, and a set of computing resources, assign tasks to resources and schedule their executions to optimize certain performance metrics. These optimizations include maximizing the number of tasks that can be processed without timing/deadline violations, minimizing the completion time of the last task (the *makespan*), minimizing the aggregate weighted completion time, and minimizing computing resources needed to accommodate the executions of a set of tasks to meet their deadlines. Well known engineering heuristics for mapping/scheduling tasks on multiple processors include First-Fit, Best-Fit, Min-Min, Max-Min, Genetic Algorithms, and Simulated Annealing. A detailed discussion of these algorithms can be found elsewhere [10].

A *periodic task* is a task that has an assigned period, is executed periodically, and must finish each execution before the start of the next period. Liu and Layland [9] derived a priority-driven algorithm for scheduling periodic tasks, called *rate monotonic (RM)*, and presented sufficient schedulability conditions for RM: a set of  $N$  tasks is guaranteed to meet their deadlines on a single processor under RM scheduling if system utilization is no greater than  $N(2^{\frac{1}{N}} - 1)$ . This lower bound, called the *minimum achievable utilization*, is a conservative but useful test for schedulability. Liu and Layland also proposed the *dynamic priority earliest-deadline-due (EDD)* algorithm and proved that it is optimal for scheduling periodic tasks on a single processor. Although both RM and EDD are optimal for scheduling periodic tasks on single processors, they fail to give provably good performance for scheduling periodic tasks on multi-processor systems [11].

In this paper, we address the resource allocation problem for stream processing on computational grids. We first examine why traditional task scheduling algorithms, including those designed for periodic tasks, are not well suited for stream processing. We then present our problem formulation and design a resource reservation system to solve the problem. Next, we outline two market-based methods for building the resource reservation system and carry out performance evaluations. Finally, we discuss future work in this area.

## II. STREAM APPLICATIONS

A stream application consists of a graph of multiple interacting and *repeatedly* executed processing units. These processing units perform particular types of processing on the incoming data streams—annotating or transforming the data

in a stream, or merging multiple streams—and publish the generated results as new streams or into persistent storage. Each repetition of the execution is triggered by the arrival of new inputs, the frequency of which can have a fixed rate, follow a specified distribution, or be completely random. Thus the conventional method of associating each task with a fixed deadline is not suitable because the exact time at which each execution can start is not known *a priori*.

In addition, stream applications have varying quality of service requirements. Some require processing and analysis to be done on-the-fly in order to enable real-time responses, with little tolerance to delay. Some have elastic deadlines, where the value realized depends on how quickly the inputs are processed. For example, the value of delivering a stock tick stream to a user with a 10-second delay is less than that with a 1-second delay, and the difference in value depends on the user. Thus the conventional method of using a priority number to indicate a task’s importance/value is not suitable.

Furthermore, rather than minimizing the number of machines used to meet the deadlines of all tasks, or the makespan of the task set, our objective is to maximize the net economic value generated by a fixed set of resources. We assume that the grid is given, and our goal is to utilize the grid for the greatest economic benefit by scheduling the processing of streams. We do not consider the possibility of generating revenue by renting grid capabilities to third parties. Given these distinctions, conventional solutions that optimize traditional metrics are not suitable for the stream environment.

Lastly, computations on streams often have substantial state; for example, a computation in a trading application maintains the state of the trade. Some operations on streams cannot be moved between computers without also moving the state associated with the operations, so pinning operations to computers in a grid is an important step in the scheduling process. This turns the optimization problem into a non-convex NP-hard integer programming problem, which we discuss later.

There are several research projects studying different aspects of stream processing, *e.g.*, STREAM [12], Borealis [13], SMILE [14], and GATES [15]. However, their approaches to resource management are fundamentally different from the one presented here, because we deal with objective functions based on economics rather than on load balancing.

### III. PROBLEM FORMULATION

We represent the computational fabric for stream processing by a graph, where the nodes represent processors and the edges represent communication channels. Each node and each edge has an associated set of parameters. Node parameters include the amount of memory, floating-point and fixed-point speeds, *etc.* Edge parameters include bandwidth and latency.

Each stream application can be represented as having three components: one or more input flows, one output flow, and a graph of interacting processing units. Figure 1 shows a pair of interacting stream applications. Each stream application has a distinct persistence interval  $[T_{start_i}, T_{end_i}]$ , where  $T_{start_i}$  is the time instant when the application starts receiving inputs

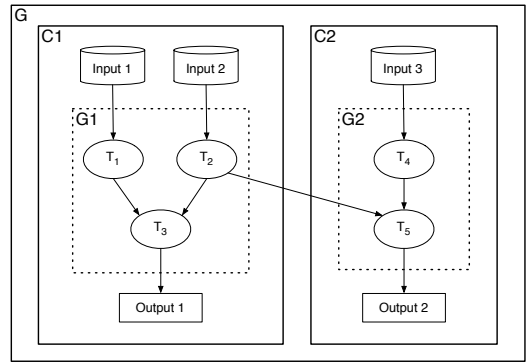


Fig. 1. Two stream applications

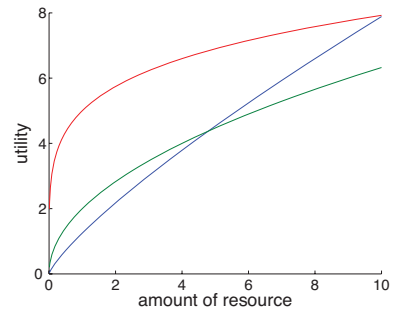


Fig. 2. Utility functions

and becomes active and  $T_{end_i}$  is the time instant when it stops receiving inputs and becomes inactive. Each stream application has a utility function  $U(r)$ , which maps an amount of resource  $r$  to the value realized by processing the inputs with that guaranteed amount of resource. For theoretical foundations about utility functions, refer to Mas-Colell and Green [16]. In most cases,  $U(r)$  is a concave nondecreasing function with parameters that depend on the application. Figure 2 shows three example utility functions. The more resource available to a stream application, the quicker it can process its input, and thus the more utility it generates.

The problem we solve is to maximize the total utility realized by all the stream applications subject to resource constraints. This is analogous to the bin packing problem with variable sized items. The computing resources correspond to the set of bins and the stream applications correspond to items whose weights depend on the variable size. The utility functions map the different sizes of each item to their corresponding weights, which are typically non-linear. In the bin packing problem, the objective is to choose the size of each item and put it into a bin, so that the total weights in all the bins are maximized. In our problem, the objective is to choose the amount of resource allocated to each stream application and assign it to a machine.

If we assume that each processing unit can be split and hosted on multiple machines, then the problem becomes a continuous convex optimization problem, which is easily

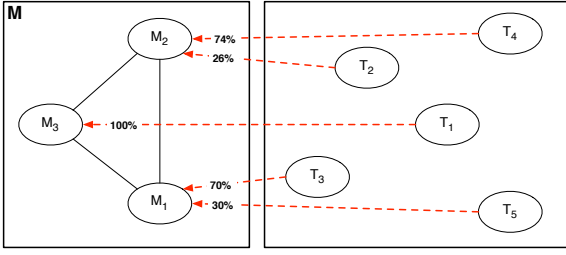


Fig. 3. System view of the solution

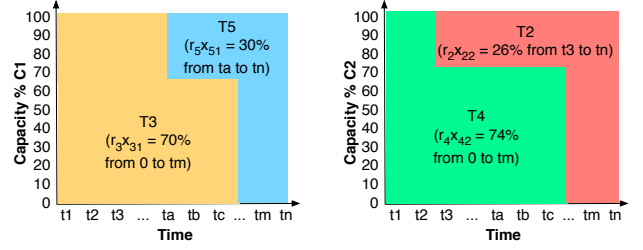


Fig. 4. Per-machine view of the solution

solved using standard optimization techniques.

$$\begin{aligned} & \max \quad \sum_{i=1}^N U_i(r_i) \\ & \text{subject to} \quad r_i \geq 0, \quad \sum_{i=1}^N r_i \leq \sum C_j \end{aligned}$$

However, the processing units often perform their computations based not only on current data but also on the history of past data; in such cases, they cannot be split and mapped to multiple machines. With this constraint, the problem becomes an NP-hard discrete programming problem. We propose using a resource reservation system to solve the problem in two steps: (1) assign each processing unit  $T_i$  to one machine  $M_j$ ; (2) reserve a certain amount of  $M_j$ 's resource  $r_{ij}$  for  $T_i$ 's execution during its existence interval. Figures 3 and 4 show a system view and a per-machine view of this solution space. Each resource reservation is of the form  $(r_i, x_{ijt}, [T_{start_i}, T_{end_i}])$ , where  $r_i$  denotes the amount of resource that is reserved for stream application  $i$ . The value of  $x_{ijt} \in \{0,1\}$  indicates whether stream  $i$  is assigned to machine  $j$  during time  $t$ .

In this paper, we investigate a simpler version of the problem, where each stream application consists of a single processing unit instead of a graph of interacting processing units, and all stream applications have the same existence interval. We propose the following problem formulation:

Let there be  $N$  stream applications, each with a concave utility function  $U_i(r_i)$ ,  $1 \leq i \leq N$ , and  $M$  machines, each with resource capacity  $C_j$ ,  $1 \leq j \leq M$ . In our system, we assume that machine capacities are large relative to an individual stream application's resource requirement. The objective is to determine the allocation of resources,  $r_i$  (the amount of resources assigned to stream  $i$  during  $i$ 's existence interval) and  $x_{ij}$  (a zero or one value indicating whether machine  $j$ 's resource is allocated to stream  $i$ ), that maximizes the total utility:

$$\begin{aligned} & \max \quad \sum_{i=1}^N U_i(r_i) \\ & \text{subject to} \quad \sum_{i=1}^N r_i * x_{ij} \leq C_j \quad \text{for all } j \in M \\ & \quad \quad \quad r_i \geq 0 \quad \quad \quad \text{for all } i \in N \\ & \quad \quad \quad x_{ij} \in \{0,1\} \quad \text{for all } i \in N, j \in M \end{aligned}$$

We can prove the NP-completeness of our problem by reduction from the Multiple Knapsack Problem (MKP). MKP involves a set of  $M$  bins with capacities  $C_1, \dots, C_M$  and a set of  $N$  items with weights  $w_1, \dots, w_N$  and values  $v_1, \dots, v_N$ . The objective is to find a feasible set of bin assignments  $x_{ij} =$

$\{0, 1\}$  ( $x_{ij} = 1$  if item  $i$  is assigned to bin  $j$ ) that maximizes the sum of values of the items assigned to the bins. That is,

$$\begin{aligned} & \max \quad \sum_{i=1}^N \sum_{j=1}^M v_i * x_{ij} \\ & \text{subject to} \quad \sum_{i=1}^N w_i * x_{ij} \leq C_j \quad \text{for all } j \in M \\ & \quad \quad \quad x_{ij} \in \{0,1\} \quad \text{for all } i \in N, j \in M \\ & \quad \quad \quad \sum_{j=1}^M x_{ij} \leq 1 \quad \text{for all } i \in N \end{aligned}$$

We omit the proof of NP-completeness for our formulation of the resource allocation problem for space reasons; it is presented in detail elsewhere [10].

Significant work has been done toward applying economic principles to resource allocation in grids. Some of this work is based on auctions [17], while some is based on competitive markets [18]. Wolski *et al.* [19] provide a good comparison of these projects. However, most of them use the conventional model: formulation of tasks and objective functions. To the best of our knowledge, no one has investigated the problem of allocating grid resources to streaming applications. In the following sections, we present two market-based methods for building resource reservation systems for streaming applications.

#### IV. MARKET-BASED RESOURCE RESERVATION SYSTEMS

We present two heuristics for building resource reservation systems using the competitive market concept from microeconomics. Both heuristics need to determine the best allocation of a machine's resource to stream applications. The best allocation can be determined by solving the utility maximization problem:

$$\begin{aligned} & \max \quad \sum_{i=1}^N U_i(r_i) \\ & \text{subject to} \quad \sum_{i=1}^N r_i x_i \leq \sum_{j=1}^M C_j \\ & \quad \quad \quad r_i \geq 0 \quad \quad \quad \text{for all } i \in N \\ & \quad \quad \quad x_i \in \{0,1\} \quad \quad \text{for all } i \in N, j \in M, \\ & \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad t \in T. \end{aligned}$$

Since the objective function  $\sum_{i=1}^N U_i(r_i)$  is concave and the constraints are linear, the problem is a convex optimization problem and can be solved using convex optimization techniques.

To avoid requiring each consumer to reveal its utility function to a centralized solver, we design a distributed method to solve the problem by using the market mechanism to solve its dual—the pricing problem. The pricing problem can be

formulated as follows: There is a single market for resource with one supplier (the machine) and  $N$  consumers (the streaming applications). The supplier has an inelastic supply function  $S = \text{capacity}$ . Each consumer has a continuous concave utility function  $U_i(r) = b_i \frac{r^{1-a_i}}{1-a_i}$ . The supplier and the consumers interact with each other and participate in the price-adjustment process of the market as follows:

- 1) The market sets an initial price.
- 2) While supply is not equal to total demand:
  - a) Each consumer reacts to price and adjusts its optimal consumption level by equating its marginal utility and the current market price; that is, it sets  $U'_i(r) = r^{-a_i} = \text{price}$ .
  - b) The supplier reacts to demand and updates the market price according to the excess demand.

The equilibrium price  $p^*$  is the price at which total demand is equal to supply. Note that the notion of price used here doesn't have the conventional monetary meaning; rather, it is a tool for determining the best allocation. At this equilibrium price each consumer has a corresponding demand  $r_i^*$ , which forms the solution to the resource allocation optimization problem.

According to general equilibrium theory in microeconomics, there exists a unique equilibrium price for this convex optimization problem at which the market clears (total demand equals total supply) and achieves maximum social welfare ( $\max \sum_{i=1}^N U_i(r_i)$ ). The convergence property of the algorithm is also guaranteed [16].

The price update process uses an interpolation method instead of a conventional step-size-based price adjustment. The interpolation method updates prices in two phases. In the first phase, it finds lower and upper bounds on the optimal price; the lower bound is a price when excess demand is greater than zero, and the upper bound is a price when excess demand is less than zero. When the bounds are found, it enters the second phase, where the new price is calculated by interpolating from the current price bounds. Experimental results show that this method results in a substantial performance improvement and cuts the number of iterations in the price adjustment process from thousands to fewer than one hundred.

#### A. Single Market Resource Reservation System

The first method uses a single market to determine the assignments of tasks to machines and the proportions of machine resources assigned to the tasks, then uses a variation of the *first-fit-decreasing-value* heuristic to assign tasks to machines. The algorithm has the following three steps:

- 1) *Local Optimization on Virtual Machine*  
Assuming a virtual machine  $M_v$  with capacity equal to the sum of the capacities of the machines in the system  $\sum C_i$ , use the local optimization algorithm to solve for the optimal allocation  $r_1 \dots r_n$ .
- 2) *Task Assignments to Machines*  
Assign stream applications to machines according to the *best-fit-decreasing-value* heuristic, where the size

of each item is  $r_i$  and the value is  $U(r_i)$ . Given the allocation  $R = (r_1^*, r_2^*, \dots, r_N^*)$ , and the corresponding utilities  $U = (U_1(r_1), U_2(r_2), \dots, U_N(r_N))$ :

- a) Sort the tasks in decreasing order of utility.
- b) Apply the best-fit-decreasing-value heuristic to assign these sorted tasks to machines.
  - i) if the next task  $T_k$  can be assigned to a machine according to its optimal allocation  $r_k^*$ , do so.
  - ii) otherwise, if  $T_k$  can not be assigned to any machine according to the optimal allocation ( $\forall j : r_k^* \geq \text{remaining-}C_j$ ), put it in the set  $\{RT\}$ .
- c) Assign the tasks in the set  $\{RT\}$  to machines using the best-fit-decreasing-value heuristic.

#### 3) Local Optimization on Individual Machines

Run the price convergence algorithm on each individual machine to optimize the allocations to the tasks assigned to that machine.

We prove that, in the worst case, the solution found by our heuristic is a 2-approximation.

*Theorem 1:* Let  $U = \sum U_i(r_i)$  be the total utility achieved by our solution and  $U_{opt}$  be the total utility achieved by the optimal allocation; then  $U \geq \frac{1}{2} U_{opt}$ .

*Proof.* Recall that our heuristic has three major steps: (1) perform the local optimization on the virtual machine; (2) assign tasks to machines according to  $r_i$ ; and (3) perform the local optimization on each real machine. The first step solves the underlying convex optimization problem exactly and finds the optimal allocation of resources to tasks assuming all machine resources are gathered into one virtual machine. Thus the total utility achieved on the first step,  $\bar{U}$ , serves as an upper bound on the optimal solution:  $\bar{U} \geq U_{opt}$ . If we can show that  $U \geq \frac{1}{2} \bar{U}$ , the proof is complete. There are two substeps in the second step: (2.1) assign tasks to machines according to  $r_i^*$  using best-fit-decreasing-value with the constraint  $r_i^* \leq \text{remaining-}C_j$ ; and (2.2) assign remaining tasks to machines according to  $r_i^*$  using best-fit-decreasing-value without the constraint  $r_i^* \leq \text{remaining-}C_j$ . We now prove that after step 2.1 the sum of utilities of the tasks assigned to a machine is at least half as much as  $\bar{U}$ . Recall that each task has a value  $U_i(r_i^*)$  and a size  $r_i^*$ . Step 2.1 first arranges tasks in decreasing order of value  $U_i(r_i^*)$ , then assigns the largest-valued task to a machine unless it doesn't fit within the remaining capacity of any machine. Let  $W_1, \dots, W_j$  be the wasted space on each machine after step 2.1. Let  $LT = \{T_k, \dots, T_x\}$  be the set of leftover tasks that can't be assigned to a machine according to their sizes  $r_i^*$ . Then

$$W = \sum W_j = \sum_{i \in \{LT\}} r_i.$$

We assumed that machine capacities were much larger than any individual task's resource requirement, thus

$$\max r_i^* < \min C_j.$$

When no remaining task can be assigned to any machine, we have:

$$\begin{aligned} \max W_j &< \min r_k^* \\ \overline{W}_j &< \overline{r}_k \\ \frac{W}{M} &< \frac{W}{|LT|} \\ |LT| &< M \end{aligned}$$

In the worst case, those  $M$  tasks are the ones with the  $(M+1)st$ ,  $(M+2)nd$ , ...,  $(2M)th$  highest values, and the sum of their values is less than or equal to the combined value of tasks  $1 \dots M$ :

$$\begin{aligned} \sum_{i=M+1}^{2M} U_i(r_i^*) &\leq \sum_{i=1}^M U_i(r_i^*) \\ \sum_{i=M+1}^{2M} U_i(r_i^*) &\leq \frac{1}{2} \overline{U} \end{aligned}$$

Therefore, after step 2.1 the total utility realized by the tasks assigned to the machines is more than half of the optimal. It is apparent that the remaining steps never decrease the total utility: step 2.2 adds remaining tasks to machines, and step 3 does a local optimization on each machine to re-balance the resources allocated to its tasks and maximize the total utilities. That is, if the original allocation found in step 2.2 is the optimal, then it is kept, otherwise resources are reallocated to achieve higher total utility. Therefore, the total utility achieved by our heuristic is at least half of the optimal.  $\square$

From the proof, it is apparent that the  $\frac{1}{2}$  lower bound is not tight. This conjecture is confirmed by experiment and simulation, as described in the next section.

### B. Multiple Market Distributed Resource Reservation System

Instead of a single market, we can model the system as multiple markets: one for each machine resource. This is analogous to the concept of multiple markets for substitutable goods in microeconomics theory. Consumers/stream applications can choose to “purchase” resources from different markets based on the price information available to them. Initially, each stream application is located in/assigned to one market (*e.g.*, based on location). When markets reach equilibrium, stream applications from higher priced machines have incentives to move to lower priced machines. Given the discrete nature of the problem, the system is not guaranteed to converge to a stable equilibrium if stream applications are allowed to move freely. Thus we enforce restrictions on when a stream application may move from one machine to another; a move is allowed if and only if it increases the sum of the total utilities of the applications on the source and destination machines. Each step of this process involves a pair of machines trying to move stream applications from one to the other so that the sum of utilities increases. This iterative monotonic process increases the total utility at each step and terminates when no stream applications can be moved from one machine to another to increase the pair-total utility. The algorithm:

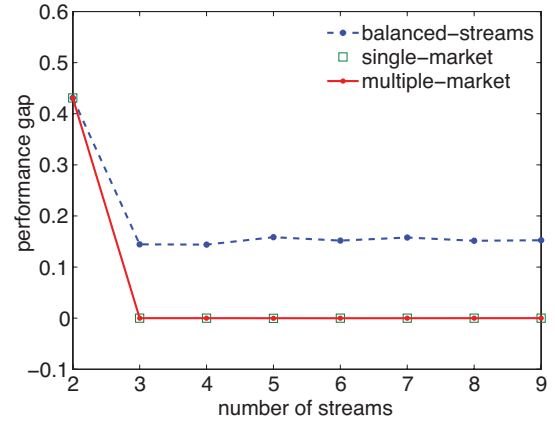


Fig. 5. Performances of the three heuristics under a heavy-tail stream distribution

- 1) Initialization: Randomly assign stream applications to balance the number of streams per machine.
- 2) Iteration:
  - a) Each machine runs a local optimization to find the equilibrium price  $p_j^*$ .
  - b) Machines send equilibrium prices to the designated central agent.
  - c) The central agent pairs up a below-average priced machine with an above-average priced machine.
  - d) The paired machines move streams from one to another to increase the sum of their utilities.
- 3) Termination: the process terminates when no single movement of a stream can increase the total utility.

## V. SIMULATIONS

In the experiments, we assume that all stream applications have utility functions of the form  $U_i(r_i) = b_i * \frac{r_i^{(1-a_i)}}{(1-a_i)}$ ,  $a_i \in (0, 1)$ ,  $b_i \in (0, \infty)$ . We choose this particular class of functions because it captures/approximates many concave functions that are typically used as utility functions [20]. To evaluate the performance of our method, we compare the total utility achieved by our method with two metrics: (1) *Upper bound*  $\overline{U}$ , the maximum total utility achieved assuming streams can split; and (2) *Base bound*  $\underline{U}$ , the total utility achieved by the naïve load-balancing heuristic, which assigns streams to balance the number of streams per machine. We compare the metrics under two distributions, a *uniform distribution* and a *heavy-tail distribution*.

In most of the figures below, we compare the performances of our heuristics and the balanced-streams heuristic by plotting the *normalized performance gap (NPG)* for each. The NPG for a heuristic is calculated as  $\frac{\overline{U}-U}{\overline{U}}$ , where  $U$  is the total utility achieved by the heuristic and  $\overline{U}$  is the performance upper bound. Thus, the NPG is a number between 0 and 1; the smaller the NPG, the better the performance.

### A. Heavy-tail Distribution

The stream processing applications follows a heavy-tail distribution if, under the optimal allocation on the virtual

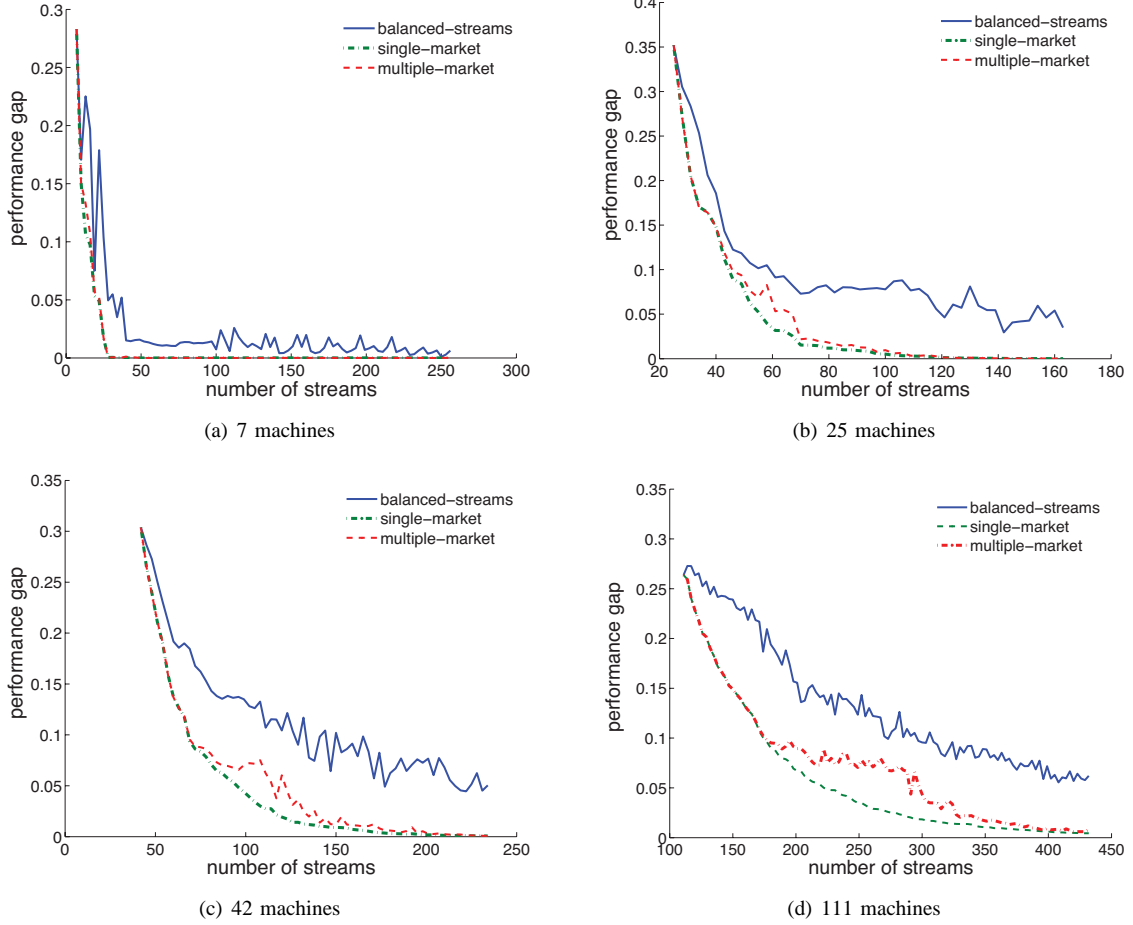


Fig. 6. Performances of the three heuristics under a uniform stream distribution, with various numbers of machines

machine, many streams have small allocations, a few streams have large allocations, and very few fall in between. This distribution is typical for flows on the Internet, and is known as the “elephants and mice” distribution. To test the performance of our heuristics under this distribution, we designed the following experiments: Assume there are two machines in the system, each with unit capacity, and  $N = \{3, 4, 5, 6, 7, 8, 9\}$  stream applications each with a utility function  $U_i(r_i) = b_i * \frac{r_i^{(1-a_i)}}{(1-a_i)}$ . Let  $M$  of the stream applications have the parameters  $a = 0.1$ ,  $b = 1$ , and  $N - M$  have the parameters  $a = 0.9$ ,  $b = 0.01$ .

Under this experiment setup, the balanced-streams heuristic has  $M^N$  different assignments. To compare the performances of the heuristics, we first compute the average performance of the balanced-streams heuristic by averaging out the sum of the total utilities achieved by all possible assignments.

As shown in Figure 5, the average performance of the balanced-streams heuristic stays at 85% of optimal, while both of our heuristics perform near 100% of optimal. The fully-decentralized multiple-market heuristic performs just as well as the single-market heuristic.

### B. Uniform Distribution

To test the performance of our methods under a uniform distribution, we designed the following experiments: Generate  $N$  stream applications each with a utility function  $U_i(r_i) = b_i * \frac{r_i^{(1-a_i)}}{(1-a_i)}$ , with  $a_i \in (0.2, 0.5)$ ,  $b_i \in (0, 1)$  chosen randomly, then run the three heuristics for each simulation. Figure 6 shows the performances of the three heuristics with 7, 25, 42 and 111 machines in the system.

The following observations are based on the simulation results:

- 1) The performances of our two market-based heuristics approach optimal quickly as the number of streaming applications increases.
- 2) The balanced-streams heuristic performs significantly worse than our market-based heuristics in all cases.
- 3) The performance gap for the balanced-streams heuristic decreases as the number of streams increases, but doesn't converge to optimal (for up to 500 streams in the system), staying from 1% to 10% below optimal.
- 4) The more machines in the system, the larger the performance gap for the balanced-streams heuristic, and the slower the convergence of the two market-based heuristics.

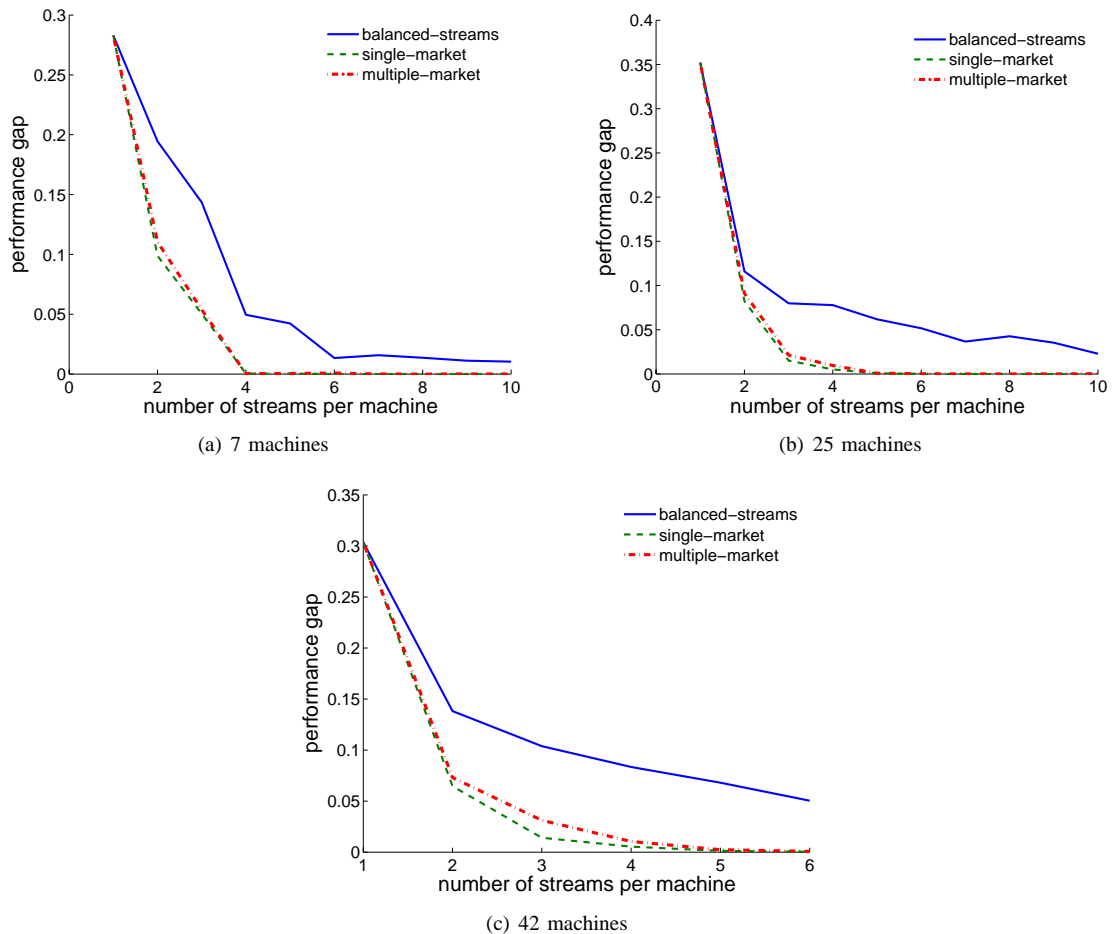


Fig. 7. The effect of streams-to-machine-ratio on performance, by number of machines

To analyze how the relative number of streams to machines affects the performance gap, we plot the performance gaps of the three heuristics with respect to the number of streams per machine. Figure 7 shows the comparisons of performance gaps among the three heuristics for various numbers of machines. As we can see from these plots, in all cases, the performance gaps of our market-based heuristics converge to 0 when there are more than 4 streams per machine while the naïve balanced-streams heuristic stays at a 1% to 4% performance gap (depending on the number of machines).

### C. Timing Analysis for the Multiple-Market Heuristic

The single-market heuristic has provably polynomial running time. However, there is no theoretical complexity bound on the performance of the multiple-market heuristic. We study the timing/computational complexity of the multiple-market heuristic empirically by measuring how long it takes for the process to converge during each simulation. We measure time by the number of rounds it takes for each simulation to complete, where each round is one attempt to move a streaming application from one machine to another (each attempt essentially entails running the local optimization step once). The average running time for the local optimization

step is 10 ms.

We plot the number of rounds against the number of streams in the system. Figure 8 shows four curves, one for each distinct number of machines in the system. One observation is that the running time increases with the number of streams in the system but does not depend on the number of machines in the system.

To study how the execution time changes as the number of streams in the system increases, we attempt to approximate it with a polynomial function. Suppose this function is of order  $x$ :  $T(n) = O(n^x)$ , where  $n$  is the number of streaming applications. We derive the following:

$$\begin{aligned} \log(T(n)) &= \log(n^x) \\ \log(T(n)) &= x \log(n) \\ x &= \frac{\log(T(n))}{\log(n)} \end{aligned}$$

Thus if we plot  $\log(\text{time})$  vs.  $\log(\text{number\_of\_streams})$  for each distinct number of machines,  $x$  can be approximated by the slope of this log-log plot. Such a plot for 7, 25 and 111 machines appears in Figure 9. It is evident from the empirical result that the time complexity of the multiple-market heuristic is approximately quadratic in the number of streams.

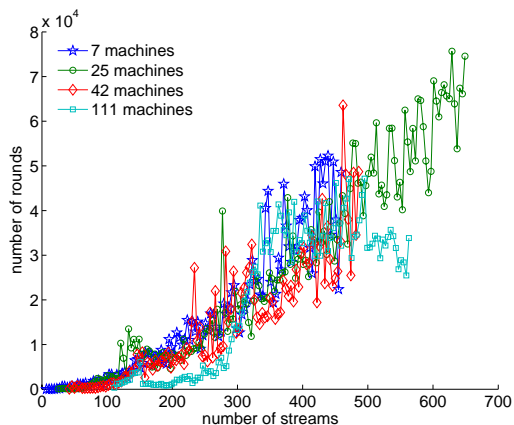


Fig. 8. Timing comparison

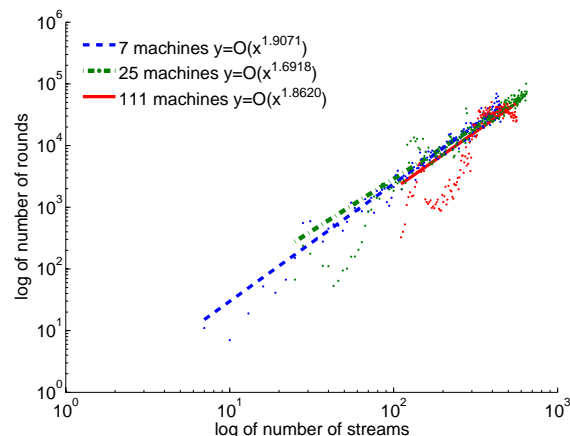


Fig. 9. Timing approximation

## VI. CONCLUSION AND FUTURE WORK

Research that integrates computer science with economics has proven useful with respect to the Internet [21], [22] and, as we have shown, is also useful in addressing the problem of scheduling stream applications on computational grids. We have investigated computational grids as infrastructures that add economic value. A new aspect of the problem studied here is that the value of an output depends on the speed with which it is generated. The design of a resource reservation system that takes this aspect into account is carried out using concepts from the literature on scheduling and economics.

We have proposed two market-based methods for building resource reservation systems, one semi-decentralized with provably polynomial running time and the other fully decentralized with empirically polynomial running time. We have proven a  $\frac{1}{2}$  lower bound on the performance of the semi-decentralized method; we have also shown by simulation that both methods provide near-optimal performance for reasonable task-to-machine ratios and outperform the naïve balanced-streams heuristic by significant amounts when the streams follow a heavy-tail distribution.

Simulation results have confirmed our conjecture that the  $\frac{1}{2}$  lower bound for the single-market-based heuristic is not tight. We are trying to formulate a proof that tightens this bound. We are also refining optimization techniques for the multiple market-based heuristic. In addition, we plan to study various extensions of this problem by relaxing existing assumptions.

## REFERENCES

- [1] A. Natrajan, M. A. Humphrey, and A. S. Grimshaw, "Grid resource management in Legion," in *Grid Resource Management: State of the Art and Future Trends*. Kluwer Academic Publishers, 2003, pp. 145–160.
- [2] K. Czajkowski, I. T. Foster, and C. Kesselman, "Resource co-allocation in computational grids," in *Eighth IEEE International Symposium on High Performance Distributed Computing (HPDC'99)*, Aug. 1999.
- [3] J. E. G. Coffman, *Computer and Job-Shop Scheduling Theory*. New York: John Wiley and Sons, 1976.
- [4] M. G. Harbour, M. H. Klein, and J. P. Lehoczky, "Timing analysis for fixed-priority scheduling of hard real-time systems," *IEEE Transactions on Software Engineering*, vol. 20, no. 1, pp. 13–28, 1994.
- [5] T. D. Braun, *et al.*, "A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems," *Journal of Parallel and Distributed Computing*, vol. 61, no. 6, pp. 810–837, 2001.
- [6] O. H. Ibarra and C. E. Kim, "Heuristic algorithms for scheduling independent tasks on nonidentical processors," *Journal of the ACM*, vol. 24, no. 2, pp. 280–289, 1977.
- [7] S. Ali, *et al.*, "Greedy heuristics for resource allocation in dynamic distributed real-time heterogeneous computing systems," in *International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'02)*, June 2002.
- [8] S. Dhall and C. Liu, "On a real-time scheduling problem," *Journal of Operations Research*, vol. 26, no. 1, pp. 127–140, Jan.-Feb. 1978.
- [9] C. Liu and J. Layland, "Scheduling algorithms for multiprogramming in a hard real time environment," *Journal of the ACM*, vol. 20, no. 1, pp. 46–61, Jan. 1973.
- [10] L. Tian, "Resource allocation in streaming environments," Master's thesis, California Institute of Technology, 2006.
- [11] C. Liu and S. K. Khall, "On a real-time scheduling problem," *Journal of Operations Research*, vol. 26, no. 1, pp. 127–140, Feb. 1978.
- [12] S. Babu and J. Widom, "Continuous queries over data streams," in *SIGMOD Record*, Sept. 2001.
- [13] D. J. Abadi, *et al.*, "The design of the Borealis stream processing engine," in *Conference on Innovative Data Systems Research*, 2005.
- [14] Y. Jin and R. Strom, "Relational subscription middleware for internet-scale," in *2nd International Workshop on Distributed Event-Based Systems (DEBS'03)*, 2003.
- [15] L. Chen, K. Reddy, and G. Agrawal, "GATES: A grid-based middleware for processing distributed data streams," 2004.
- [16] A. Mas-Colell, M. D. Whinston, and J. R. Green, *Microeconomic Theory*. New York: Oxford University Press, 1995.
- [17] C. A. Waldspurger, T. Hogg, B. A. Huberman, J. O. Kephart, and W. S. Stornetta, "Spawn: A distributed computational economy," *Software Engineering*, vol. 18, no. 2, pp. 103–117, 1992.
- [18] D. Abramson, R. Buyya, and J. Giddy, "A computational economy for grid computing and its implementation in the Nimrod-G resource broker," 2002. [Online]. Available: [citeseer.ist.psu.edu/abramson02computational.html](http://citeseer.ist.psu.edu/abramson02computational.html)
- [19] R. Wolski, J. S. Plank, J. Brevik, and T. Bryan, "Analyzing market-based resource allocation strategies for the computational Grid," *The International Journal of High Performance Computing Applications*, vol. 15, no. 3, pp. 258–281, Fall 2001.
- [20] J. Mo and J. Walrand, "Fair end-to-end window-based congestion control," *IEEE/ACM Transactions on Networking*, vol. 8, no. 5, pp. 556–567, Oct. 2000.
- [21] S. Shenker, "Fundamental design issues for the future Internet," *IEEE Journal on Selected Areas in Communication*, vol. 13, no. 7, Sept. 1995.
- [22] A. Tang, J. Wang, and S. Low, "Counter-intuitive behaviors in networks under end-to-end control," *IEEE/ACM Transactions on Networking*, vol. 14, no. 2, Apr. 2006.