Coordinated Workload Scheduling: An Emerging Application Domain for Mechanism Design

Elie Krevat
Carnegie Mellon University
ekrevat@cs.cmu.edu

Abstract

The process of distributing and coordinating parallel workloads across many computation or storage nodes is a hard scheduling optimization problem. Furthermore, in distributed systems where nodes require monetary incentive-based payment schemes to schedule workloads efficiently and fairly, or a central coordinator might not have enough information to make the best system-wide decisions, framing scheduling problems in the context of a monetary incentive-based mechanism design can lead to globally efficient and fair solutions while nodes make individually-optimizing decisions. We motivate the application of mechanism design in two areas of coordinated workload scheduling. We first apply an auction that uses a Vickrey-Clarke-Groves mechanism to improve the scheduling process in a free distribution setting, where many nodes may service any workload. We then study an environment with a fixed distribution setting, where a fixed set of nodes is assigned to each workload, and show that the problem of coordinating even a single timeslice of these workloads across all workloads and nodes in the system is hard. While the solution to workload coordination in this fixed distribution setting is still an open research problem, we frame the problem in the context of a mechanism design by computing a number of derived properties of schedule allocations, showing how these properties may be used in a payment scheme that gives the nodes incentives for fair and efficient schedules, and define the goals of a mechanism that distributes the assessment of possible schedule allocations over many nodes in the system.

1 Introduction

As distributed systems of nodes performing computation or storage tasks become larger and more complex, the burden of scheduling synchronized workloads across nodes in an efficient manner becomes an increasingly important problem. Synchronized communication in these distributed environments requires the coordination of many individual nodes that could each be servicing possibly many clients and workloads. Each node also ideally provides some quality of service (QoS) guarantees to the workloads for which it is responsible, providing fairness when allocating resources across workloads.

When synchronized tasks are uncoordinated in time over many nodes, and a workload requires responses from many nodes before it can proceed, then the performance of the system approximates that of the slowest node. One particular environment where this problem surfaces is in cluster-based storage systems, where data blocks are striped over many storage servers such that a read request for a block cannot complete until all queried storage servers respond with their data fragments. Other environments where this problem occurs is in distributed systems with heterogeneous collections of servers, where a task is decomposed into many distributed tasks that are all completed in parallel for better performance. As we show in this paper, the problem of determining an optimal coordinated schedule even for one timeslice in a time-sharing distributed system is hard even with a simple restricted environment model.

Further complicating the task of scheduling in distributed systems is the emergence of new computing environments with monetary incentives, where the owners of the workloads and the owners of the nodes are separate entities and a central scheduler may not have all the necessary information to make efficient decisions. For instance, an ambitious goal of Grid Computing [2] is to allow individuals to sell spare computation cycles of their personal desktops. In these environments where individual nodes may lie about their performance and capabilities, the system

must impose a mechanism to enforce cooperation and enable more efficient load distribution and scheduling decisions.

The purpose of this paper is to motivate the application of mechanism design in distributed scheduling coordination schemes, both for existing distributed systems and new emerging systems with monetary incentive-based models. Through an analysis of a free workload distribution setting, where workloads can be freely assigned to many nodes, we demonstrate that the problem of load distribution can be solved with an auction using the classic Vickrey-Clarke-Groves (VCG) mechanism [7, 6]. However, in a fixed workload distribution setting such as cluster-based storage, where workloads are assigned to a fixed set of nodes, we show that the problem of scheduling workloads coordinated in time is a hard optimization problem without a clear mechanism design solution, although larger goals still apply to maximize the efficiency and fairness of the system while nodes make self-optimizing local decisions.

The outline for the rest of this paper is as follows. Section 2 describes the importance of coordinating workloads in the motivating example environments of cluster-based storage and emerging systems with monetary incentives. Section 3 argues for the application of mechanism design to solve scheduling problems in these environments and defines our basic modeling approach. The next major sections then study two important scheduling problems from different workload distribution environments, *free workload distribution* and *fixed workload distribution*. Section 4 provides a revenue model for the free workload distribution setting and a VCG-based mechanism design to schedule workloads on the most efficient subset of nodes. Section 5 shows why an optimal solution for coordinating workloads is hard even in a simple restricted setting, provides a revenue model for the fixed workload distribution setting, defines one possible payment scheme that gives incentives to nodes to schedule workloads efficiently and fairly in coordination, and describes the goals and avenues of further research for a full mechanism design solution. Section 6 presents our conclusions.

2 Background and Motivation

This section describes modern distributed systems, such as cluster-based storage systems, that provide a shared computing environment where coordinating the timeslices of workloads across nodes is an important way to improve the efficiency and fairness of schedule allocations. Additional problems scheduling workloads in coordination occur in emerging systems with monetary incentive-based payment models, where the owners of workloads pay the system or the individual nodes for computation and storage tasks, and the central scheduler cannot always obtain full and truthful information from the nodes in the system. We define two important scheduling problems in different workload distribution environments and argue for the application of mechanism design to solve coordination problems in these environments.

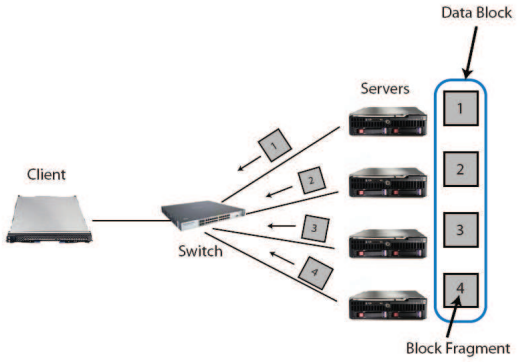


Figure 1: A cluster-based storage environment stripes data across groups of servers for reliability and performance. Since the performance of a synchronized read request is limited by the time it takes for the slowest server to respond, this example motivates the need for coordinated workload scheduling

2.1 Coordinated Workloads

Workloads in distributed systems may be run in parallel across many nodes performing computation and storage tasks. Clients may run many different workloads at once, and nodes may be shared by many workloads. Because nodes divide their time fairly among the workloads that they service, important issues arise with scheduling workloads to the right distribution of nodes, and scheduling the timeslices on nodes so that each workload is coordinated in time.

Synchronized workloads are defined as workloads that send out parallel requests and are then blocked until all its requests have been processed and results have been returned. These workloads are particularly sensitive to slow or uncoordinated nodes. They are also sensitive to long periods of latency while waiting for requests to complete. Therefore, the goal of any

efficient system when scheduling synchronized workloads is to minimize the latency and the variation in response times of nodes that are assigned these workloads.

Nodes that participate in time-sharing become unsynchronized for many reasons. First, they might calculate different lengths of quanta for their timeslices. Also, the groups of workloads serviced by each node varies, and the number of timeslices in a cycle of workloads will be different for a node that services more workloads.

One particular motivating example of this is cluster-based storage [1]. As depicted in Figure 1, storage servers serve data for many different workloads, and data blocks are striped into fragments across groups of servers for reliability and performance. In the simple example for when a client makes a synchronized read request for a single data block, that single data block request actually corresponds to many block fragment requests, sent in parallel to an assigned set of storage servers. Only when the client receives all data fragments for that block will it initiate the next request.

Coordinated workloads become increasingly important when it is the goal of the system to provide QoS guarantees of throughput performance. Projects like Argon [8] provide *performance insulation* for a single storage server, where a workload or the system can specify a bounds on the performance hit that a workload will take when sharing a machine, compared to the performance that it would have received with a dedicated machine. The idea is that n workloads should be able to share a machine and each achieve better than $\frac{1}{n}$ of their standalone performance by applying strategies such as dedicated caches and longer timeslice quantas that batch requests together and achieve a minimum level of performance.

2.2 Monetary Incentives

Many emerging systems with monetary incentive-based models add additional complexities for scheduling coordinated workloads. These environments occur when nodes are owned by separate entities than clients, and nodes require compensation to run the clients' workloads. A central scheduler in this type of system may not have all the information necessary to make the best scheduling decisions, and they may have incorrect information if nodes lie about their performance capabilities in order to make more money.

Typical workloads in this space are massively parallelized. For instance, weather forecasting simulations that are computationally intensive might be split up into many distributed calculations, and then individual results are merged together to get the final system-wide result. Oil companies also run computationally intensive simulations on seismic measurements to estimate the amount of oil reserves available before spending millions of dollars attempting to drill in a location.

Many other distributed systems are incorporating models where they might sell their resources to interested parties. For instance, supercomputing centers might sell time on their system if they have spare capacity. Different companies might have an agreement to share the same compute cluster, and within a single company different organizations might share infrastructure to reduce management costs and take advantage of economies of scale.

One particularly ambitious project is Grid Computing [2], where one day it might be possible for individual desktop machines connected to the Internet to sell their spare computation cycles or storage capacity to others. Grid computing also introduces many other issues such as the lack of one central administrative domain which are outside the scope of this paper, but interesting when considering how the owners of workloads will find the best set of nodes to run on, and how money would be transferred and agreements enforced in such a domain.

3 Modeling Approach

We argue for the application of mechanism design to achieve global metrics of efficiency and fairness for scheduling coordinated workloads. We also make a number of assumptions in our environment to simplify our analysis, but these assumptions can be relaxed in future research so that our results can apply to a larger number of other environments. Only when we have the right model for our environment, where nodes make revenue-optimizing local decisions, can we begin to approach the problem of finding the right type of auction mechanisms and payment schemes that optimize social welfare and, in turn, system efficiency.

3.1 Applying Mechanism Design

The application of mechanism design provides certain desired properties, such as the efficiency for some

total value metric or revenue for the agents. Mechanism design applies especially well for situations when central coordinators lack per-node information such as the performance and load of a node in the system. Without this knowledge, and if the nodes cannot be trusted to truthfully provide this information, scheduling decisions may not be done efficiently.

When applying global QoS metrics to an existing system, an effective mechanism will help to ensure that the motivation of the nodes corresponds to the motivations of the system. For instance, fairness and efficiency is not always the goals of nodes in the system, but a mechanism that rewards nodes for schedules with these properties allows the optimization of global metrics to occur at the individual nodes.

Additionally, it might be much more efficient in a limited time period to query the nodes for their best schedule allocations, feed their responses into a general mechanism, and interpret thes responses in a way that minimizes the time of computation at a central coordinating node. When the task of finding an optimal schedule is especially hard and computationally intensive, as we will show applies to scheduling the next timeslice of coordinated workloads in Section 5, this distribution of computation becomes even more important.

In our distributed system environment, any mechanism design will also need to incorporate a payment scheme for its players. This payment scheme will determine how money is transferred from the users of the system, who must pay for scheduling their workloads, to the nodes of the system that should be compensated for their role in servicing workloads. If workload owners are willing to pay more for a more efficient schedule, then a payment scheme might also make similar incentives for payments to the nodes to provide additional compensation in certain situations.

3.2 Environment Assumptions

Our distributed computing environment contains a number of different factors which can complicate our analysis. Therefore, we present a simplified system model that bounds the number and variety of clients and workloads and allows each node's scheduler to communicate and respond to information in parallel with any computation or storage tasks.

First, we assume that each client is responsible for one and only one workload and that there are a constant number W of workloads available in the system at any time. The actual system consists of S nodes, and we assume that the computing capacity of the system is large enough to handle all workloads.

Our mechanism assumes that there are a constant set of workloads to be scheduled and no additional workloads are added to the system. However, since in reality workloads are changing all the time, if we restrict ourselves to auction mechanism that only schedule one or two timeslices into the future then we will be able to better respond to changing workload conditions.

We assume in our environment that all workloads are of equal priority, and that each node's goal (perhaps driven by monetary incentives) should be to provide on average a fair allocation of resources across the workloads for which it is responsible. Nodes also attempt to minimize latency when switching between workloads at the granularity of a time quanta. Therefore, if *workload* i (w_i) has just been scheduled for *node* x (n_x), and n_x is responsible for just two workloads, w_i and w_j, then n_x would strongly prefer to schedule w_j in the next time quanta.

Additionally, we assume that nodes can communicate within the system and determine their schedules without conflicting with any other computation or storage tasks for the currently running workload. This assumption allows us to abstract away the overhead of an active scheduler on other tasks, and allows us to concentrate on the benefits of such fine-grained control.

3.3 Basic Revenue Model

We impose a revenue model on our system where nodes are paid an amount for scheduling a synchronous workload, as long as that workload is scheduled in a coordinated manner across all other nodes that service that workload. Nodes are compensated for each time quanta that it schedules for a synchronous workload in coordination. Uncoordinated workloads are not important in our system. During periods when none of their workloads can be scheduled in coordination, nodes may work on other unsynchronized workloads or perform other necessary background tasks, but payments for these other tasks will not be considered in our analysis.

A model that pays each node individually rather than the system as a whole does not generally apply to our motivating example of a storage system. However, this model might apply to some specific systems, where nodes may be loaned by different people or organizations that require compensation. It is also consistent with each node's individual scheduling pro-

cess in these systems, where a node makes localized scheduling decisions based on its current load and quality of service guarantees. Therefore, the preferences of any one node does not always match that of other nodes, and an environment where nodes try to deliver on its local quality of service guarantees can be fitted with monetary payment schemes.

While we have described the basic revenue model for this environment, additional revenue model details will be given for different individual settings in the remainder of this paper. the actual payment scheme details are also left up to the mechanism design. We propose one possible payment scheme in Section 5.4.

3.4 Studying Distribution Settings

We will now consider two primary workload distribution settings, and within each of these settings we focus on a representative scheduling problem for coordinated workloads. In a *free workload distribution* setting, workloads can be freely assigned to any subset of nodes in the system. Embarrassingly parallel computation problems are good example workloads fit for this setting. In contrast, a *fixed workload distribution* setting may only assign workloads to a fixed set of nodes in the system. Data fetching applications in cluster-based storage systems are good example workloads fit for this setting.

4 Free Distribution

In the free workload distribution setting, a representative problem that we solve with mechanism design is the problem of determining the best set of nodes to assign to each workload.

4.1 Revenue Model

We propose an environment with free workload distribution and a revenue model that compensates nodes after a task is completed depending on the level of service achieved. Payment may either come from an authoritative node in the system, or directly from the owners of the workloads. The amount of payment should depend on many factors, including the speed of response and the number of requests and computations that are committed to the workload per timeslice. Clients pay only based on the level of service they receive and don't discriminate based on the type of node. Interestingly, the exact payment model to the nodes will not affect our solution, so long as nodes are compensated more for better performance and that amount is easily assessed by the nodes before agreeing to take a workload. This model also assumes that nodes can effectively estimate their free resources.

The clients (i.e., owners of the workloads) may also pay a fixed cost to the central scheduler as compensation for using the system in the first place. The central scheduler plays a key role in determining which set of M nodes to assign the workload. Workloads want the best and fastest nodes possible. Coordinated workloads especially want not only the fastest nodes, but nodes that are very close to the same performance level, since their performance is anyway dependent on the slowest node.

The central scheduler in this environment is lacking important information from the nodes that is not easily obtainable. For instance, nodes may not be trusted to release their true current load or speed because they would otherwise be assigned many workloads where they would be paid some amount but provide on average a poor response.

4.2 Mechanism Design

The important system goal in this setting is to assign workloads to the nodes that will respond fastest, and in order to determine the fastest nodes, it must be up to the central scheduler to design a mechanism that is incentive compatible and will induce the nodes to reveal their true characteristics.

A mechanism design that elegantly solves this system goal is the application of an auction with a Vickrey-Clarke-Groves (VCG) mechanism. This auction would run as follows:

First, the central scheduler would run a classic sealed-bid auction to decide which M nodes to assign to a particular workload. The scheduler might run this auction on just a single workload at a time in a first-in-first-out queueing approach. Alternatively, the scheduler might bundle many of these workloads together and run a combinatorial auction. Either way, it would be up to the individual nodes to assess their expected revenue for running a job based on their actual performance characteristics, and based on this number, supply their valuation of the workload to the central scheduler.

After receiving all N nodes' valuations for the workload, we apply a VCG mechanism. The classic VCG mechanism says to remove some winning player X and run an additional auction iteration, where the

payment of player X is the difference in the social welfare of both auctions or equivalently the opportunity cost lost by others for having participated in the auction. In our mechanism, if we assume that the second auction is done immediately after the first with a negligible time difference, then the amounts of the bids of each node will not change. Therefore, the price that each node n_x of the M winning nodes pays is equivalent to the difference between the bid of n_x and the $M+1$ highest bid.

An additional modification of the auction is possible if the system does not want the central scheduler to make money. To spread the payments of any node over the rest of the participating nodes, a central scheduler may normalize the payments by a constant amount. However, if we assume that there are many nodes, then we can also assume that the difference between any of the M winning bids and the $M+1$ highest bid is very small, so payments in our system will also be very small.

4.3 Mechanism Results

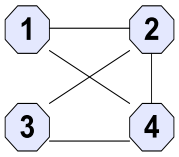
Our VCG-based mechanism has a number of desirable properties. First and foremost, it provides incentive compatibility to our domain. Nodes have no incentive to lie, since if they over-report their valuation for a workload then they'll still only be paid its true performance valuation. This is a very strong result and the main reason why we used the VCG-based mechanism in the first place.

Our mechanism also achieves global efficiency. The central scheduler can now correctly determine the best and fastest group of nodes to schedule a workload on, because each node is now trustworthy when assessing the valuation of a workload, If we assume that communication costs and the assessment of the valuation of a workload is constant, then the complexity of our auction for one workload is $\theta(N+M)$.

4.4 Related Work

It's important to note that some work in this area of task allocation has been done previously by Nisan and Ronen [5], where k tasks need to be allocated to n agents. However their environment is different than ours, they do not treat tasks as distributed, and they consider only the combinatorial auction case for which an optimal solution requires exponential time, but a polynomial time approximation algorithm exists.

WkId	Nodes
1	A,B
2	B,C
3	C,D
4	A,B,D



Node	WkIds
A	1,4
B	1,2,4
C	2,3
D	3,4

Figure 2: Example table and diagram for determining dependencies between workloads, used to show that determining an optimal fully-coordinated schedule for even one timeslice is hard.

5 Fixed Distribution

In the fixed workload distribution setting, a representative problem that we frame as a problem for mechanism design is that of coordinating the timeslices of workloads so that if a schedule specifies that a node n_x is assigned a workload w_i, then all other nodes that service w_i are also assigned that workload in the schedule.

This section begins by showing that the scheduling coordination problem is hard even in a simple restricted setting where we use a myopic approach for scheduling allocations only for the next subsequent timeslice. We then continue by filling in the final details of the revenue model, most of which were described earlier in Section 3.3. After clarifying the revenue model, we frame the context for a mechanism design and describe the goals of such a system.

One of the integral parts of any mechanism design is a payment scheme for its players to provide incentives for the players to perform locally in a way that will maximize global system efficiency. We propose one such payment scheme, where payments are made based on the quality of schedule allocations. Nodes then compute a number of simple and derived properties of total schedule allocations (i.e., allocations of a larger number of consecutive timeslices) to assess their efficiency and fairness. These properties are used to determine the actual payments for each node.

We conclude this section with a final discussion of the open research topics in this area.

5.1 Optimal Coordination is Hard

The scheduling problem for finding the optimal set of workloads to run in coordination for even a single timeslice is hard, and with further research may even be NP-complete. We demonstrate its hardness

by showing how a graph can be formed of dependencies with other workloads, and that the coordination problem can be reduced to the Max Independent Set problem which is NP-complete, however we have not yet determined a reduction from the Maximum Independent Set problem to the coordination problem, so we have not yet determined the exact hardness of this problem.

For every node n_x that services a workload w_i, then w_i has a dependency edge to all other workloads serviced by x_i. Referring to Figure 2, which shows the same information in two tables and graph form, we see that Workload 1 has a dependency with workloads 2 and 4 (and trivially itself) because Workload 1 is serviced by nodes A and B, and the union of the sets of workloads on nodes A and B is $(1, 2, 4)$, Taking this example to the end, we find that the largest group of workloads that can be run in one timestep is 2, specifically the set $(1, 3)$, which is also the maximum independent set of the resulting dependency graph.

Even if the workload coordination problem is not NP-complete, the blow-up of dependencies makes this problem hard to manage.

5.2 Revenue Model

We propose an environment with a fixed workload distribution and a revenue model that compensates nodes after a task is completed depending on the achieved level of service. While in the free distribution setting a node was paid either by the system or by the owners of the workloads, in the fixed distribution setting we give full control of the payment to the system, and the actual payment details are still left to the mechanism.

We also assume that each workload is distributed across a random subset of N nodes, and a single node cannot be assigned more than M workloads. For our storage system example, data blocks might be striped over any number of servers depending on the level of erasure coding, but many commercial storage products use a restricted range depending on reliability and performance requirements. For example, Panasas stripes blocks over RAID groups of 8 to 11 servers [3].

The central scheduler in this case wants a quick resolution of all workload requests, in a globally efficient and fair manner, but the nodes need monetary incentives to schedule workloads with these properties. Uncoordinated workloads are not important in this settings and therefore are not compensated.

5.3 Mechanism Design Goals

The important system goals in this setting is to enforce coordination of timeslices for each workload, achieve a fair distribution of resources between all workloads, and achieve an efficient schedule that tries to minimize wasted cycles when there are workloads waiting to be scheduled in the queue.

An auction mechanism in our system should assign the nodes of the system to be the players of the auction. For the motivating example of a cluster-style storage system, the players would be the storage servers. While it might seem more intuitive at first to have the clients or the workloads be the players in our auction, the goal of our mechanism is to improve the efficiency and fairness of the system through auction-based methods while using a limited computing model [4]. If the clients or workloads were players, then they would be competing for resources on the nodes, but one very rich client could buy all the resources of the system (as in a costly computing model) and prevent fairness across workloads. Therefore, an auction with the clients as players would serve a very different purpose.

The good to be auctioned is a schedule allocation of coordinated workloads to nodes, whereby each node must comply with the coordinated allocation once that allocation has been decided. Given a set of workloads, and the set of nodes that service each workload, a valid single-quanta allocation assigns exactly one workload to each node such that every scheduled workload is *fully coordinated* across all the nodes that service that workload. By fully coordinated, we mean that if a workload w_i is assigned to node n_x, then all of the other $N - 1$ nodes that service that workload must also be assigned w_i. A valid total allocation assigns a series of single-quanta allocations over a longer time period. We assume that the total allocation time period is long enough for a node to service a sufficiently large representative sample requests for all the workloads, but short enough so that the time it takes to assess and auction all possible valid allocations is not prohibitive.

A node n_x might not be assigned any workload, possibly because other nodes that share a subset of workloads with n_x are assigned to a disjoint subset of workloads. If a node is not assigned any workload in the allocation then it may work on other workloads that do not require coordination, or perform other background management tasks. While the problem of finding an optimal schedule allocation even for a single time quanta was shown to be hard, an auction

mechanism should still ensure that no obvious pareto improvements can be made to a scheduled allocation to include additional coordinated workloads for unallocated nodes.

Depending on the particular requirements and capabilities of the system, an auction mechanism may choose to use either the total allocation or the single quanta allocation. A mechanism for assessing and solving for a good allocation in each case will probably depend on a number of heuristics. While it is easier to assess long term fairness and efficiency properties of a total allocation, searching even with these heuristics over an exponential number of possible total schedule allocations may not be feasible. Additionally, the set of workloads in the system might be changing at each time quanta, in which case a single quanta allocation auction would avoid delays in scheduling these new workloads until the end of a total allocation period. However, in a single quanta allocation nodes will need to adjust their valuations for the next schedule based on past allocations, and we would expect that the final resulting schedules over a longer period of time would not be as good as for the case of the total allocation schedule.

Therefore the tradeoff between a single quanta allocation auction and a total allocation auction is the ability of the system to make best effort computations for efficient allocations while responding quickly to changes in the set of workloads to be scheduled, and the ability to better assess the efficiency and fairness of allocations over the long term. The final mechanism will probably fit somewhere between these two extremes of a single and total allocation and find some efficient smaller bundle of sequential time quanta that can be efficiently auctioned together.

5.4 Payment Scheme

We propose one possible payment scheme where a node is paid a maximum of P credits for each time quanta that it schedules for a synchronous workload in coordination. A node receives no credits for scheduling a synchronous workload in an uncoordinated manner. For every cycle of time quanta that goes by that a particular workload has not been scheduled in a coordinated manner, the payment for that workload decreases a small constant discount amount c to provide further incentive to schedule a workload sooner rather than later. A cycle of time is defined as the amount of time necessary to schedule all the workloads on a node in the best case that they are all scheduled consecutively without any unscheduled quanta.

By itself, decrementing the payment of one workload is not enough to provide incentive for a node to schedule every workload in a fair manner. One might consider the situation where one workload has not been scheduled for a long period of time and therefore a node is further discouraged from scheduling it any more, since its payment would be smaller than the payment of other possible workloads. In order to correct this situation and further motivate each node to schedule workloads fairly, a node is fined F credits if it starves a workload by not scheduling it synchronously longer than a period of time quanta Q_{thr}, where Q_{thr} is the threshold value after which a fine is incurred, and F is substantially larger than P.

5.5 Properties of Schedule Allocations

A basic quanta allocation a_{quanta} assigns workloads to servers for a single quanta time slot. In an environment where workloads are steady and unchanging, then schedule allocations can be grouped together over a longer sequence of time quanta, which will minimize the number of auctions necessary to coordinate servers. We refer to this total allocation as a_{total}, and our primary analysis in this section covers this particular type of allocation because longer-term properties and statistics can be more easily assessed in this setting. Alternative auctions for finer-grained quanta allocations are also discussed.

Given a total schedule allocation a_{tot} of workloads to servers over a longer period of time, some important basic per-server properties are required. First, according to the assumptions of our environment in Section 3.2 there are W total workloads in the entire system and Q total time quanta possible to be scheduled. The identifiers for all workloads i assigned to a node n_x is also determined and stored in the set S_x.

Using these basic per-node properties, the first important derived property of a total schedule allocation is the number of coordinated time quanta successfully scheduled for all workloads identified in the set S_x of a node n_x, without any distinction yet for which workloads are scheduled. We refer to the number of time quanta devoted to each workload w_i as the *quanta count* q_i. Due to the requirements of coordination in our system, the local q_i values are equivalent to the global values for each workload. We then compute the *total quanta count* Q_x for each node n_x as the sum of all q_i important to a node:

$$Q_x = \sum_{\forall i \epsilon S_x} q_i \quad (1)$$

The second important derived property of a total allocation is the *schedule distance* $d_{i,k}$, defined for each workload w_i identified in the set S_x for node n_x and corresponding to the kth instance of that workload scheduled in the allocation. From this schedule distance we calculate two other properties, the *average schedule distance* $\bar{d_i}$ and the *maximum schedule distance* $\widehat{d_i}$. If the allocation does not allocate any time quanta to a workload then the schedule distance is defined to be equal to $Q + 1$, or one unit greater than the total possible number of time quanta in the allocation. The schedule distance of a workload's first instance of a scheduled time quanta is defined to be equal to the position of that time quanta in the sequence of all time quanta in the total allocation. For example, if w_i was allocated in the first time quanta, followed by an unallocated quanta, and then w_j, then $d_{i,1}$ would equal 1, and $d_{j,1}$ would equal 3.

The average schedule distance $\bar{d_i}$ is defined for a workload w_i as follows:

$$\bar{d_i} = \frac{1}{q_i} \sum_{k=1}^{q_i} d_{i,k} \quad (2)$$

Similarly, the maximum schedule distance $\widehat{d_i}$ is defined for a workload w_i as follows:

$$\widehat{d_i} = \frac{1}{q_i} \max_{k=1}^{q_i} d_{i,k} \quad (3)$$

These properties of schedule distance correspond to notions of fairness for the total allocation. Even more fine-grained properties are available such as considering the variance of the schedule distances, however the average and maximum schedule distances are sufficiently descriptive for our environment, since nodes care most about on average how frequently they schedule a specific workload and to make sure that any one serviced workload is not starved long enough to warrant a fine. Future work may choose to incorporate additional measures.

Similar to our calculation for Q_x, we then compute the *total average schedule distance* $\bar{D_x}$ of all $\bar{d_i}$ important to a node n_x as follows:

$$\bar{D_x} = \sum_{\forall i \epsilon S_x} \bar{d_i} \quad (4)$$

And we also compute the *total maximum schedule distance* $\widehat{D_x}$ of all $\widehat{d_i}$ important to a node n_x as follows:

$$\widehat{D_x} = \max_{\forall i \epsilon S_x} \widehat{d_i} \quad (5)$$

5.6 Determining Payments for Schedule Allocations

In order to assess the value of payments to each node for total schedule allocations, we first examine the best- and worst-case values for allocation properties and then map these results into a payment function.

The best-case value for the total quanta count Q_x of a node n_x is $Q_x = Q$, meaning that every single time quanta available to n_x was successfully scheduled with a coordinated workload. On the other hand, a node without any coordinated workloads scheduled receives a valuation of 0 (and not a negative valuation) since the cost of running a server is an initial sunk cost that does not factor into our consideration.

Similarly, the best-case value for the maximum schedule distance D_x of a node n_x is $D_x = |S_x|$, meaning that the node has a perfect round robin scheduling of all its workloads in a coordinated manner, scheduled in the same order each cycle. Note that this best-case situation could only occur when the total quanta count is also at its best-case maximum of Q. On the other hand, a node would be less and less happy as the maximum schedule distance D_x gets larger because it would approach the threshold Q_{thr} for which a workload is considered starved. When this threshold is exceeded, a fine of F would offset any residual valuation of the total schedule allocation. We include the negative impact of this fine directly into the valuation function.

Based on the above observations, and using the constant discount amount c to refer to the amount deducted from the payment P for each workload for every quanta time that a workload isn't scheduled, the payment function p_x for each node n_x is calculated as follows:

$$p_x(a) = P*Q_x - c(\bar{D_x} - 1) - F*\max(\min(\widehat{D_x} - Q_{thr}, 1), 0) \quad (6)$$

5.7 Discussion

The coordinated workload scheduling problem, as defined in this paper, is a hard optimization problem. One approach to a solution is to solve the optimization problem at one central location and then broadcast the results to all the nodes. However, if possi-

ble, a better model for solving this optimization is to spread the computation across each individual node, to have each node determine its best allocations that maximize payments based on a payment scheme, send its valuations to the central scheduler, and have the central scheduler use these values to determine the best global allocation.

Traditional auction mechanisms provide certain desired properties that when applied correctly will maximize some total value metric or revenue for the agents. Therefore we would like to apply some auction mechanism, where individual nodes can determine (through heuristics) the best allocations to bid on, and therefore reduce the optimization problem of determining a good schedule allocation into a more general auction form that achieves our goals. In effect, this would be a general mechanism to spread out computation across nodes and agree on a consensus for all nodes.

In our fixed distribution setting, however, existing mechanisms such as the Vickrey-Clarke-Groves mechanism do not readily apply. Given the environment and context of our problem as described previously, let us speculate what a Vickrey-Clarke-Groves mechanism might look like in order to show its shortcomings. First, an auction would be performed with all the nodes and total allocations to determine the maximum payments possible across the system. Each node solves only for the allocation which maximizes its own payments (which as we defined before also corresponds to the node's utility). But with the current payment scheme all of the fixed nodes are paid equally, and if any one node is then removed from the computation it would be impossible to schedule that same workload in coordination, and the same effect would occur for every node that has been assigned a workload.

5.8 Future Work

Future work should first examine more closely what a good model is for a utility function of the nodes in the fixed distribution setting. The utility for each node classically depends both on the payments that are received into the system mechanism and by the owners of workloads and subsequent payments from the system mechanism to each of the servers (including any assessed fines) based on the role of each server in achieving the right QoS guarantees and efficiently serving and scheduling workload requests. We have proposed one such payment scheme above that would be used in part by the node to value a particular allocation. It is still up to the mechanism design to figure out what kinds of payments are necessary for the nodes to pay to the system (or are normalized away to spread the wealth).

However, beyond the incentive of payments, it is not readily apparant what additional concepts should be incorporated into a valuation function for the nodes. After all, if it is the job of an auction to come up with the payment schemes that provide incentives to the nodes to schedule jobs efficiently and fairly, then what additional utility do these nodes get for a particular allocation besides the payment?

Future work in this area will also continue to build on the system models and results which we have proposed in this paper, solve for the exact hardness of the coordinated workload problem, and research a mechanism design that achieves the right goals for the system, or otherwise determines that an auction-based design is not appropriate for this particular problem and instead proposes a pure optimization problem that may be computed by the central scheduler.

6 Conclusions

Distributed systems offer a rich set of scheduling problems that may be well suited for the application of mechanism design. Emerging systems such as Grid Computing are especially suited for designs that promote incentive-compatibility in a setting where nodes otherwise have no real incentives to truthfully reveal their performance characteristics. It is up to a mechanism design in these areas to achieve the goals of better global performance, efficiency, and fairness by motivating each node to attain these goals locally.

We have specifically analyzed two different distribution settings to come up with a model and application for mechanism design. The free workload assignment setting fit nicely into the domain of an auction with a VCG mechanism, and gave clients the ability to find the best set of nodes to run their workload. The fixed workload assignment setting's problem of scheduling coordinated workloads is still an open research problem, but we have made good progress in defining the problem in the context of a revenue model, showing that the basic optimization problem is hard with many dependencies between workloads, and demonstrating a payment function that used derived allocation properties and gave nodes incentives to favor allocations that are fair and efficient.

We are encouraged by the initial results in finding applications for mechanism design in coordinated

workload scheduling, and hope to have provided motivation for further research in this and other areas of scheduling and distributed computing.

References

[1] M. Abd-El-Malek, W. V. C. II, C. Cranor, G. R. Ganger, J. Hendricks, A. J. Klosterman, M. Mesnier, M. Prasad, B. Salmon, R. R. Sambasivan, S. Sinnamohideen, J. D. Strunk, E. Thereska, M. Wachs, and J. J. Wylie. Ursa minor: Versatile cluster-based storage. In *FAST*, 2005.

[2] C. Kesselman and I. Foster. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers, November 1998.

[3] E. Krevat, V. Vasudevan, A. Phanishayee, D. G. Andersen, G. R. Ganger, G. A. Gibson, and S. Seshan. On application-level approaches to avoiding tcp throughput collapse in cluster-based storage systems. In *Proceedings of the 2nd Petascale Data Storage Workshop, Supercomputing*, November 2007.

[4] K. Larson. *Mechanism design for computationally limited agents*. PhD thesis, Pittsburgh, PA, USA, 2004. Chair-Tuomas Sandholm.

[5] N. Nisan and A. Ronen. Algorithmic mechanism design. In *Proceedings of the 31st ACM Symposium on Theory of Computing*, pages 129–140.

[6] N. Nisan and A. Ronen. Computationally feasible vcg mechanisms. In *EC '00: Proceedings of the 2nd ACM conference on Electronic commerce*, pages 242–252, New York, NY, USA, 2000. ACM.

[7] T. W. Sandholm. Limitations of the Vickrey auction in computational multiagent systems. In V. Lesser, editor, *Proceedings of the First International Conference on Multi-Agent Systems*. MIT Press, 1995.

[8] M. Wachs, M. Abd-El-Malek, E. Thereska, and G. R. Ganger. Argon: Performance insulation for shared storage servers. In *Proceedings of the 5th USENIX Conference on file and Storage Technologies*, February 2007.