Decentralized Prioritization-Based Management Systems for Distributed Computing

Per-Olov Östberg and Erik Elmroth
Dept. of Computing Science, Umeå University, SE-901 87 Umeå, Sweden
{p-o, elmroth}@cs.umu.se

***Abstract*—Fairshare scheduling is an established technique to provide user-level differentiation in management of capacity consumption in high-performance and grid computing scheduler systems. In this paper we extend on a state-of-the-art approach to decentralized grid fairshare and propose a generalized model for construction of decentralized prioritization-based management systems. The approach is based on (re)formulation of control problems as prioritization problems, and a proposed framework for computationally efficient decentralized priority calculation. The model is presented along with a discussion of application of decentralized management systems in distributed computing environments that outlines selected use cases and illustrates key trade-off behaviors of the proposed model.**

I. Introduction

Distributed computing environments, such as high-performance, high-throughput, grid, and cloud computing environments consist of highly complex systems with increasing degrees of heterogeneity and needs of automation. In this work we address construction of decentralized management systems for distributed computing infrastructures, and present a model for decentralized prioritization systems along with a discussion of application of this model in current distributed computing environments.

Distributed systems are systems where multiple nodes (processes, programs, or computers) collectively contribute to common system goals, e.g., perform individual parts of large computations or coordinate the use of shared resources. Construction of scalable distributed systems is a challenging problem that typically involves exploitation of parallelism and asynchronous programming techniques. Decentralization is a distributed systems technique focused on elimination of centralized coordination of nodes that has several benefits for scaling of distributed systems, including, e.g., reduced (control) information flow, elimination of system-wide single points of failures, reduced system and node load sensitivity, and facilitation of better models for data parallelism and autonomous node operation in systems.

Formulation of computationally efficient decentralized distributed systems is challenging. As nodes in distributed systems typically operate on incomplete views of system information, and there are latencies in data updates, distributed systems must encompass mechanisms that provide robustness in node operation to compensate for data and control degradation. To support decentralization, underlying problems addressed by systems as well as system algorithm formulations must support decentralized operation. In general, the more complex a distributed software is (i.e. the more external dependencies, and complex interactions and functionality it has), the harder it is to formulate decentralized algorithms and data structures.

Prioritization is a technique in scheduling that assigns quantifiable priorities to entries in a queue, priorities that can be used to determine sort order or otherwise influence processing of the queue. Prioritization mechanisms are in distributed computing infrastructures found in, e.g., systems for queuing, brokering, and scheduling of computational tasks, as well as in storage and quota enforcement systems. A common characteristic of these systems is that components containing prioritization mechanisms (e.g., coordination and management components) are tightly coupled to other functionality of the system, imposing limitations on system scalability and flexibility, and making it hard to distribute these components.

As many distributed computing control and management systems can be formulated as scheduling and prioritization problems, and prioritization is a decentralizable problem [21], we propose an approach to improving eScience environment scalability through use of decentralized prioritization techniques. More specifically, we propose a model for decentralized prioritization-based management systems with direct applications in distributed computing infrastructures. With a starting point in recent work on grid fairshare scheduling, we derive a generalized model for construction of decentralized management systems, and outline use cases and trade-offs in use of priority-based management in eScience environments. The rest of the paper is structured as follows: Section II gives a brief survey of related work, Section III outlines the proposed model for decentralized management systems, and Section IV discusses formulation and application of prioritization-based management in eScience environments. Section V concludes the paper, followed by acknowledgements and references.

II. Background and Related Work

The proposed decentralization model traces its roots to an operating system scheduling strategy called fairshare scheduling [14] where the core idea is to base context switching on process owners (users) rather than processes. This allows differentiation of processes and give users with few processes the same amount of CPU time (i.e. their ”fair share”) as users with many processes. The concept of fairshare is later extended to differentiation of users, formulated in quota systems that allow prioritization of users and definition of fairness in terms of users receiving run time proportional to quota allocations.

978-0-7695-5083-1/13 $25.00 © 2013 IEEE
DOI 10.1109/eScience.2013.44

In high-performance computing (HPC) environments, fairshare scheduling is extended to cluster level where jobs are scheduled to compute nodes in orders based on multiple prioritization factors (a strategy in common use in cluster schedulers such as SLURM [32] and Maui [13]). Here fairshare prioritization is a centralized process that operates on comparisons of complete views of historical usage data, and fairshare is considered one scheduling prioritization factor among many. For grid computing environments (that here somewhat simplified can be seen as distributed HPC environments), this is reformulated as a decentralized process that isolates the prioritization component in scheduling [11] and later extended to a standalone mechanism for fairshare scheduling [21]. The contributions of this work is differentiated from these prior publications in that this addresses a generalization of both the prioritization model as well as the applications of the model. The Aequus systems [21] can be seen as a special case of the generalized model here proposed, designed for the use cases of grid and high-performance computing fairshare scheduling.

Alternative algorithm formulations for distributed fairshare exist [19] and include, e.g., solutions that relax the hierarchical constraints of the policy model [17], formulations that measure fairness in terms of quality of service (e.g., amount of job scheduling requests accepted [18] or adhering to [10] and minimizing missed [7] task deadlines), as well as game-theoretic approaches that allow local optimizations of scheduling decisions [25]. For a comprehensive study of share-based prioritization and scheduling systems, see [8].

Alternative decentralized approaches to distributed fairshare also exist and include, e.g., solutions for desktop grid systems [6] and bio-inspired algorithms for decentralized load balancing in networks [22]. Decentralization and fair resource scheduling is also an area of interest in related fields, such as wireless systems using non-uniform rate assignment algorithms [23] and studies of information requirements in theoretical economy [5]. Decentralization in data sharing is also a field that in recent years has received increased attention for, e.g., decentralized storage systems such as Cassandra [20].

Biologically inspired systems that model complex systems in terms of simple, local, and individual-based interactions hold great promise for decentralization, and have been studied for a variety of purposes, including e.g., decentralized proportional fair scheduling in networks [22] and bio-inspired principles for evolution of software traits for autonomic self-organization algorithms [9]. However, while bio-inspired systems often naturally encompass decentralization, formulation of goal-orientation and computationally efficient structures are often hard as evolution tends to result in systems specialized to the specific use cases for which the systems are evolved.

Within control theory, there is an established subfield for decentralized control [26], [29], where control is modeled using differential equations (often posed as matrix problems). While this work essentially targets control problems, it is based on distributed systems theory and models distributed architectures as systems devoid of centralized control. To avoid confusion and differentiate this approach from control theory, the more general term management systems is here used to denote the proposed prioritization-based control systems.

III. Decentralized Prioritization-Based Management

There are many types of distributed computing infrastructures in production use today, e.g., High-Performance Computing (HPC) infrastructures, High-Throughput Computing (HTC) networks, federated grid computing infrastructures, and public Infrastructure-as-a-Service (IaaS) cloud computing offerings. While there are fundamental differences in the design and use cases of these different types of infrastructures, they do share some common properties including, e.g., being constructed using large, complex software stacks and middlewares, and that they are experiencing increasing levels of complexity and heterogeneity in both software and hardware.

The scale and complexity of current distributed software systems are identified as resulting in an existing and increasing need for decentralized and self-managing autonomic control systems [16]. The proposed approach addresses construction of automated and scalable decentralized management systems and is based on modeling of complex producer-consumer dynamics in terms of generalized computational capacity, formulation of control systems as prioritization problems, and design of software systems in a decentralized framework composed of three main types of functionality: policy specification and resolution, data distribution, and priority calculation.

A. Distributed Capacity Producer-Consumer Dynamics

Distributed computing infrastructures contain complex producer-consumer dynamics, and management systems need to be able to capture multiprovider and multitenancy scenarios, as well as high dimensionality in the various types of capacity produced (e.g., network, I/O, compute, and storage capacity) and indirect quality of service metrics such as security, overhead, and varied service levels. In addition, large-scale distributed infrastructures also have scalability and performance requirements that necessitate decentralized formulations of control and management systems. The vision of this work entails decentralized distributed management systems where nodes operate autonomously and allow distributed computing actors (infrastructure providers, end-user organizations, control systems, etc.) to steer production and consumption of capacity towards target policies without centralized coordination.

The following traits are identified as desirable for generalized decentralized management systems:

- Ability to model different types of computational capacity in complex producer-consumer settings.
- Ability to dynamically adapt to rapid and frequent changes in both target formulation and system behavior.
- Ability to robustly adjust management system behavior without external coordination (self-organization).
- Capture of hierarchical formulation of target functions that model hierarchies in user organizations.
- Goal-oriented, decentralized, and computationally efficient operation without any kind of centralized control.

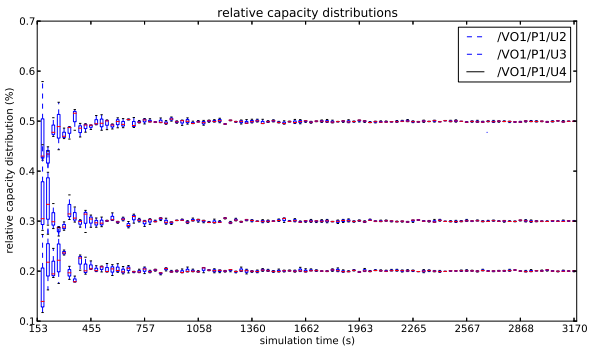

Fig. 1. Convergence of capacity consumption to predefined capacity allocations within a subgroup of a virtual organization. Boxplots displaying relative capacity distributions over time for multiple (10) simulations.

B. Prioritization-Based Management

The definition of management systems used here details systems that manage, direct, or regulate the behavior of other devices or systems[1]. In control theory, controlled systems are typically guided towards intended system behavior using target functions that express some approximation of optimal system behavior. To formulate management systems for the complex producer-consumer problems outlined in the previous section, control is here reformulated as a prioritization problem:

In what order should system tasks be performed to achieve the best approximation of optimal system behavior?

To formulate target functions for optimal system behavior, metrics or heuristics to measure system behavior are needed. In order to construct prioritization-based management systems, we express target functions for desired system behavior in relative terms, e.g., in ratios of how much of predefined capacity quotas that have been consumed, and measure system behavior deviations (from target functions) to establish priority orders for system actions. Measurements of system behavior are based on historical capacity consumption data fed back into management systems in closed feedback loops. The system action priority order, or adjustments of system actions based on priority recommendations, are used to regulate system behavior. To capture modeling of hierarchy in systems (e.g., differentiation of users at different levels in hierarchical organizations), we define target functions in tree structures where optimal capacity distributions are defined in relative shares within subgroups (i.e. percentages between nodes on the same level of the tree, see right-hand side of Figure 2).

Using this approach, any problem that can be formulated in terms of ranking a set of independent entities against each other (using a metric or heuristic that captures meaningful differentiation of the entities), can be posed as a prioritization-based control problem. Management systems constructed using decentralized prioritization can from a high-level perspective be viewed as automated controllers for capacity-sharing systems, where a control framework is used to automatically divide capacity between entities using predefined capacity allocations. An example of this is detailed in Figure 1, which illustrates accumulated capacity consumption (CPU usage) over time for subgroup P_1 of the allocation tree in Figure 2. Capacity consumption is automatically steered toward predefined capacity allocations by the management system.

In the proposed model, capacity metrics can be direct, e.g., amounts of gigabytes of storage currently used, or complex, e.g., linear combinations of the amount of seconds of CPU time, watts of power, and megabits of bandwidth capacity a virtual machine has consumed in the last week. As the capacity metrics in the proposed model are normalized into shares (percentages within subgroups), and hierarchically aggregated (including child subgroups in parent capacity consumption aggregations), the exact metrics used to meter capacity consumption are not explicitly used in priority calculations (illustrated in Figure 3). Instead, what is expressed in the model is a ratio between the relative (within subgroup) desired capacity consumption (i.e. policy target consumption or desired behavior) and the relative actual capacity consumption (as measured by the metric). It is this ratio between desired and measured behavior that forms the basis for prioritization of system actions and control of system behavior.

In summary, the proposed model poses control problems as prioritization problems and defines control target functions in relative shares of produced capacity. The computational model is applicable to all control problems where it is possible to formulate capacity metrics that capture meaningful differentiation of capacity consumption, and relies on prioritization of system actions (or alterations of system actions based on prioritization) to regulate system behavior.

C. Decentralization of Prioritization-Based Management

The definition of decentralization used here entails systems to be formulated without centralized control structures or external coordination for internal operation of nodes. With this definition, all nodes in a decentralized system operate autonomously. Note that this does not mean that systems do not exchange control information, nodes may still require and rely on external information in algorithms, there is just no central coordination of the distribution of that information. While prioritization in itself is an expressive way to formulate management systems, the real benefit of using prioritization-based management systems lies in that prioritization can be addressed using decentralized control structures. In the proposed model, all main functionalities of such management systems can be decentralized:

1) Target function specification and resolution: As illustrated in Figure 2, hierarchical policy target trees are dynamically assembled at management system nodes from policy target tree components. Specification of tree components can be recursively delegated to actors (e.g., virtual organization, project, or end-user group administrators), and multiple policy

[1]By this definition, the systems discussed here can be seen as control systems. As this work is not based on control theory however, the more general term management system is used to avoid confusion.

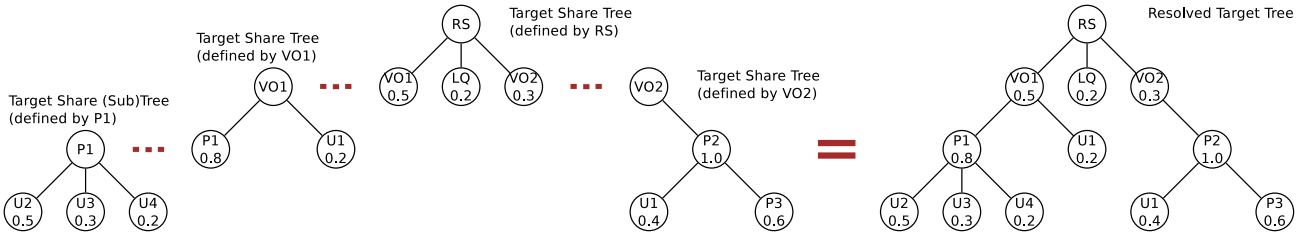

Fig. 2. Hierarchical allocation policy target tree constructed via recursive delegation, allowing policy actors to participate in policy specification. Allocation target trees are dynamically resolved at policy enforcement points (typically capacity production and management sites, here denoted RS for resource site).

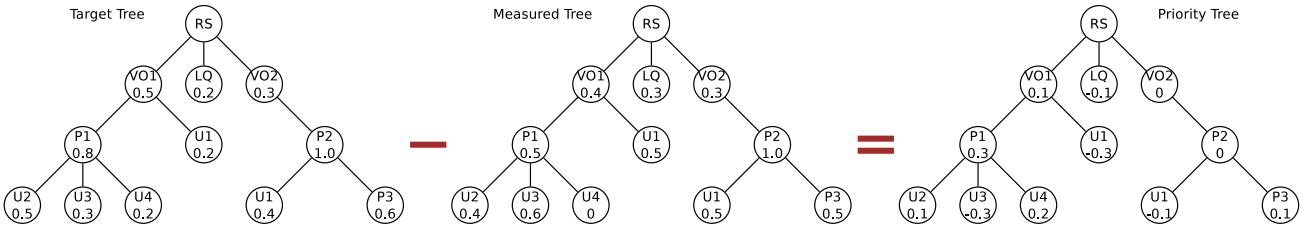

Fig. 3. Priority calculations are done through node-wise application of priority operators (in the example subtraction) on policy target trees and measured capacity consumption trees. To facilitate calculation, capacity consumption data are mapped to policy target tree structures and normalized on each node level.

trees can be recombined to form node-specific policy target trees without external synchronization.

2) Data distribution: As management data is fed back into the system in control loops, all nodes that actively participate in management systems both produce and consume control (capacity consumption) data. While use of the structure of the policy target trees can be used to segment data distribution systems for performance reasons, it is not a requirement for the model to operate. In fact, all capacity consumption data (which is the only information propagated between nodes) can be shared using techniques such as distributed hash tables, distributed databases, or shared file systems.

3) Priority calculation: As can be seen in the (somewhat simplified) illustration of the priority calculation process in Figure 3, priority calculation depends only on data from the policy resolution (policy target tree) and data distribution (capacity consumption data structured into a measured tree) processes. As these processes are decentralized, priority calculation can be addressed using data-parallel algorithms that operate independently of other nodes in the system. However, while priority calculation algorithms can be unaware that they operate on distributed data, it can in certain applications (e.g., real-time systems) be advisable to ensure that the algorithms used are capable of compensating for noise factors such as data update latencies. As priority calculations are decentralized (and produce all information required to regulate system behaviors), management systems based on decentralized prioritization systems will inherently be decentralized themselves.

D. Decentralized Prioritization-Based Management

The framework for the proposed model is constituted by three main processes that run concurrently throughout the system: policy resolution (setting and resolution of target behavior for managed systems), data distribution (filtering and propagation of capacity consumption data throughout management systems), and priority calculation (determination of priority orders for system adjustments). As an overview, policy resolution is a continuous process that detects and responds to changes in policy target formulations, and can be viewed as a configuration monitor for management systems. Data distribution and priority calculation are in the model separated, which allows individual treatment of the two problems and a conceptual view of the system as a computational pipeline (illustrated in figures 4 and 5 respectively).

1) Policy Specification: Policy specification is a decentralized process driven by the actors of the distributed computing environments. Intended goal behaviors for managed systems are defined in policy target tree components, where the owner of the entity modeled by the tree component (e.g., a project administrator for a scientific project) specifies how capacity allocated to the entity should be subdivided between organizational dependents of the entity (e.g., end-users contributing to a project). Capacity allocations are, as illustrated in Figure 2, specified in relative shares (percentages) and can thus be mapped to any metric the management system measures capacity in (e.g., gigabytes of storage, megabits of network throughput, or amount of virtual machines assigned to a system in a cloud infrastructure). As policy target trees are modeled hierarchically, policy target specification can be delegated to actors in dependent organizations (in the example of Figure 2, virtual organization VO_1 delegates specification of capacity shares for project P_1 to the administrators of P_1).

2) Policy Resolution: Policy resolution is the process of resolving policy target trees from tree components, and is performed dynamically at the site of policy enactment (i.e. where a prioritization-based management system is hosted). Note that policy enactment points may, but are not necessarily

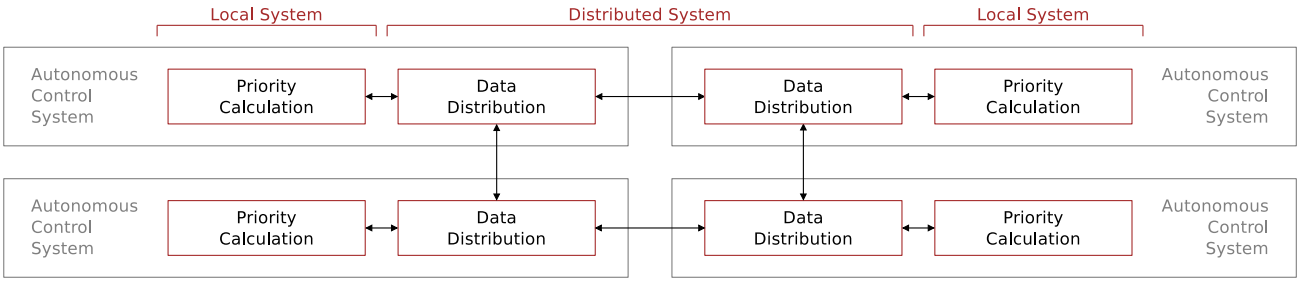

Fig. 4. Architectural overview of the proposed model. Data distribution is separated from priority calculation and abstracts data filtering and distributed system mechanics. Priority calculation is a local process that employs adaptive methods and self-adjusting algorithms to compensate for data loss.

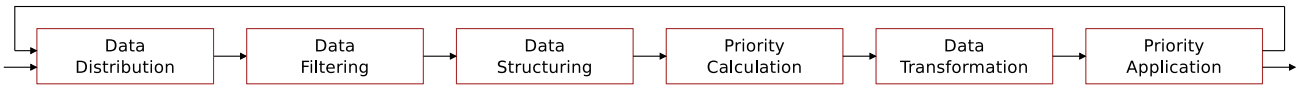

Fig. 5. Information flow. Note that policy resolution is a continuous concurrent process that affects all parts of the priority calculation chain in the model.

limited to, be cohosted at capacity production sites. In HPC scenarios it may be logical to perform policy resolution close to capacity production (as schedulers are natural prioritization points), but in network management scenarios routers may perform individual prioritization (and hence policy resolution). As policy target trees can map multiple subtrees as dependents of a root node, and individual nodes in decentralized management systems can autonomously decide what target trees to enact (and hence what policies to enforce), policy resolution should typically be performed close to priority calculation.

The remainder of this section discusses the priority calculation pipeline illustrated in Figure 5. For more details on the priority calculation algorithm in a specific use case, see [21].

3) Data Distribution: Data distribution is the process of propagating capacity consumption data throughout decentralized management systems. As nodes in management systems perform system action prioritizations, and thus steer capacity consumption, nodes act as both producers and consumers of control (capacity consumption) data. Data can be propagated using different policies (e.g., all raw data everywhere, most important data first, etc.) as well as using different types of data propagation mechanisms (e.g., peer-to-peer topologies, delta-compressed data, etc.). While exact formulation of data distribution topologies and mechanisms are outside the scope of this work, it is noted that separation of data distribution and priority calculation allows separate and more direct treatment of the data distribution problem, and that design of data distribution patterns and topologies may be advantageously based on the hierarchical structure of policy target trees.

4) Data Filtering: As data is shared among nodes in decentralized prioritization-based management systems, filtering is performed to reduce data volumes and modulate impact of capacity consumption data. In data distribution, data is filtered based on source and destination, as well as on the data needs of nodes, to minimize bandwidth requirements for updates. At nodes, and as part of the priority calculation pipeline, data filtering is applied for structuring and filtering purposes, e.g., in structuring capacity consumption data in time window histograms and by use of low-pass filtering techniques to reduce oscillations in system adjustments. In certain scenarios, data modulation functions may also be used to alter the impact of capacity consumption data, e.g., in HPC fairshare it is common to use usage decay functions to emphasize the relative importance of more recent jobs.

5) Data Structuring: As preparation for priority calculation, consumption data are mapped from time windows to tree structures inherited from policy target trees. To facilitate computation, consumption trees are also subjected to a normalization process where all nodes are normalized relative to their siblings in the tree, i.e. producing a normalized tree expressing relative shares of capacity consumption on all levels of the tree. The goal of this preparation is to encapsulate all prioritization information in tree-shaped data structures, and allow priority calculation algorithms to be separated from data distribution issues and generically formulated to operate on tree structures using priority operator functions.

6) Priority Calculation: Priority calculation is in the model designed in terms of algorithms that employ priority operator functions, a type of kernel function operating on nodes in trees, to define comparison of policy target functions to capacity consumption data. As priority operators are at the core of the framework, operator design and shape heavily affects the performance of the system. In essence, priority operators define a dot product transfer function between the system balance ratio (i.e. the ratio between policy target and capacity consumption) and the prioritization values used in systems management. Priority calculation produces a priority tree, a single data structure containing all prioritization information of the system. Note that prioritization semantics for management systems are based as much on how priority calculation results are (later) used as how they are produced.

7) Data Transformation: After priority calculation, priority vectors are extracted from the priority tree based on paths in the tree (that uniquely represent the entities modeled by

the policy target tree). Direct comparison of priority vectors corresponds to a top-down parse order for the priority tree and establishes a comparison semantic that enforces isolation of subgroups in comparisons and enables comparison of entities from different parts of the tree. To facilitate computation, vectors are typically transformed to a format suitable for the prioritization application, e.g., projected onto scalar values for comparisons with other factors in linear combinations of prioritization factors. Note that transformation of vectors typically introduce a truncation of the value space of the vectors and that prioritization information may be lost in this process if care is not taken.

8) Priority Application: As a final step of the computation pipeline of the model, priority vectors (or some derivation thereof) are used to perform rankings of the entities modeled in the policy target tree. Rankings may be complete (comparing all entities in the tree) or partial (comparing an active subset of the entities), and may for performance reasons also be prepared in advance of ranking requests (i.e. precomputed and cached). Assigning high priorities to the nodes farthest from balance between their target function and capacity consumption (from the under-utilization side) establishes a self-adjusting prioritization semantic (called least-favored-first) that is noise tolerant and self-stabilizing even in the case where not all prioritization recommendations are enacted [21].

IV. Applications of Decentralized Management

As stated by Brewer's CAP theorem, a data-sharing distributed system can not simultaneously guarantee consistency, availability, and partition tolerance for the entire system [3]. However, by selective partitioning of systems, it is possible to achieve acceptable trade-offs between these three desirable properties for data-sharing distributed systems [4], and the impact of these limitations can be further mitigated by careful design towards desired operational qualities of systems [1].

The proposed model addresses a subtask very common to distributed systems (prioritization) and places focus on decentralization for robustness and scalability reasons. The fundamental limitations outlined by the CAP theorem are addressed through partitioning schemes for prioritization information that allow individual nodes of a system to operate autonomously while (using self-adjusting techniques) still robustly contribute to system goals under non-ideal conditions.

In distributed systems there exists trade-offs between what active (control) and passive (data) information is propagated through the system. The proposed model displaces all control to decentralized nodes and propagates only passive information between nodes. The primary scalability limitation of the proposed model is that historical resource consumption data must be communicated to participating nodes, which is the price the model pays for decentralizing control. However, the amount of data needed to be distributed can be reduced through use of data partitioning and distribution schemes that exploit the hierarchical structure of allocation policy trees.

Key to the proposed approach is formulation of control problems as prioritization problems, which takes different expressions in different contexts. Here we discuss selected potential use cases for prioritization-based management, and explore application and implications of use of the proposed model for decentralized management systems.

A. HPC Batch System Queue Filtering

Traditional HPC scheduling systems such as SLURM [32] and Maui [13] organize incoming job requests in batch queues. Depending on configuration, batch queues can be further segmented into active and inactive queues, or prioritization queues where jobs are scheduled subject to multi-objective scheduling criteria such as limitations of the number of concurrent jobs for a particular user or fairshare quota limitations for user groups. Many state-of-the-art production scheduler systems exclusively use reactive scheduling algorithms that cyclically poll jobs from queues when resources become available, and match jobs to resources on a one-to-one job-resource basis.

Alternative use of prioritization information, e.g., fairshare job rankings based on user prioritization, could allow scheduler systems to be extended to filter batch job queues to exclude jobs unlikely to be scheduled from active queues. Reduction of queue sizes is likely to facilitate improvements in scheduling, e.g., increased quality in job-resource matching (task placement is a hard multi-objective optimization problem, reduced queue sizes reduce the complexity of the optimization problem), and could also allow construction of new types of scheduler technology, e.g., inclusion of proactive meta-scheduler plan-ahead techniques in cluster schedulers.

In such scenarios, policy target trees would be formulated to express user and user group capacity quotas, which in prioritization would be matched against historical capacity consumption data to establish job scheduling orders. By convention in HPC deployments, usage decay functions (sometimes in conjunction with low-pass filtering) are used to modulate usage data before calculation of fairshare priorities. A key concern for application of the proposed prioritization model in cluster schedulers is to produce prioritization factors that can be combined with other scheduling factors in schedulers, typically by rendering scalar values that are expressed in linear combinations of scheduling factors in job prioritization. For use of prioritization information to cull batch queues this is not a concern given that jobs are filtered solely on a single prioritization factor.

B. Decentralized Storage System Management

Decentralized prioritization is a natural extension in distributed storage scenarios. For scalability and performance, it is highly desirable for individual parts of large-scale data storage systems to be able to make autonomous decisions on how to enact organization-wide storage policies, such as quota enforcement rules and data management actions. In addition to direct control of storage capacity, decentralized prioritization also finds application in storage scenarios for heuristics-based decision support, e.g., in detection and management of storage resource failures, or in data migration and storage consolidation scenarios.

Storage capacity metrics used in policy target trees can here include not only direct quantities such as storage space in gigabytes, but also differentiation of indirect capacities such as I/O and transfer capabilities (higher prioritized users get more bandwidth or faster access), storage lifetime measurements (higher priorities store data longer and lower priorities get data deleted earlier in space reclamation scenarios), redundancies in storage (more copies stored of higher prioritized data), as well as localization of storage (quantifying distance to data in latency sensitive scenarios, higher prioritized users get data stored closer or on sites with higher transmission capacity).

In distributed infrastructure scenarios, decentralized prioritization can also play many roles on the client side. Third party tools and cloud computing toolkits such as the Globus Online [12] Reliable File Transfer utility and the Nimbus Toolkit [15] raise abstraction levels and allow scientists to work more efficiently in distributed computing environments. While tools such as these are changing how distributed computing environments are built and used, there is still a great need for automation and intelligent management of computations and data in many fields of science.

Decentralized prioritization-based management systems can facilitate construction of user-centered tools and programming models for management of distributed scientific data. For distributed storage systems for example, decentralized prioritization can play a natural role in tools that automatically balance load between different storage systems, or use heuristics to automatically search and retrieve data according to some optimization criteria, e.g., minimization of transfer time or access cost. In the face of the current scales and growth rates of scientific data, tools that provide automation subject to intelligent and intuitively formulated (priority-based) policies are expected to be increasingly important in eScience.

C. Network Traffic Control

In network control, use of decentralized prioritization could be used to allow routers to autonomously determine prioritization order for packets, to, e.g., decide packet drop order in congestion scenarios and determine packet filtering prioritization to enforce differentiated bandwidth quality of service in network flow. In network traffic control, policy target trees express shares in terms of network metrics, e.g., bandwidth consumption quotas. Similarly, capacity consumption can be metered both locally and globally, and consumption data propagated to nodes in condensed formats.

A key challenge in prioritization-based network control lies in exchanging capacity consumption information between nodes without consuming too much bandwidth. Additionally, the real-time nature of network traffic control applications necessitates the use of adaptive local feedback in management nodes, i.e. mechanisms where local capacity consumption data is stored and used locally, and updates are sent to other nodes in bandwidth-conserving bursts. An implication of the use of policy target tree structures in priority calculations is that the proposed model can be implemented to precompute and cache priority information, significantly reducing response time and improving throughput of prioritizations.

The decentralized nature of packet-switching networks is one of the key enablers for the growth of the Internet, a great illustration of the scalability power of decentralization. Lack of centralized control allows nodes to dynamically join and leave decentralized management systems, and facilitates a model for autonomous prioritization applicable to a vast range of control problems. Real-time management of packet-switching networks poses many challenges where the decentralized nature of the proposed model maps well to the topology of the underlying infrastructure, e.g., in local control of routers, or in mapping of overlay networks to physical networks.

D. Meta-Scheduling Quality of Service

In addition to using prioritization to construct direct control systems, prioritization information can also be used indirectly to augment or alter the functionality of other systems. For example, use of prioritization information as feedback into meta-schedulers allows construction of more advanced scheduler systems capable of providing improved end-user quality of service in grid environments (an example illustrated in [31], where the technique is demonstrated to result in meta-scheduler systems that distribute not only compute capacity but also system availability and scheduling failures proportional to capacity quota allocations).

In such combined systems, target functions need to be specified in metrics compatible with the functionality of the integrated systems. In the example of [31], policy target trees are formulated in capacity share quotas derived from grid virtual organization quota systems. In grid meta-scheduler systems that proactively plan resource allocations for a finite future time window, prioritization information can be used to not only influence resource allocation plans, but also to imply an order for updating resource allocation plans when needed (i.e. replanning resource allocations for lower prioritized jobs before altering plans for higher prioritized jobs).

In a similar manner, jobs from higher prioritized users can also be assigned to more trustworthy or better performing resources in high-throughput computing environments. Note that even in the absence of explicit quota specifications (i.e. lack of policy target trees), prioritization systems can here be used to ensure that all end-users are equally likely to suffer scheduling failures (in the example transforming a first-come-first-serve semantic to a round-robin semantic for failures by implicitly constructing share trees where all users have the same $1/n$ shares).

E. Reservation and Reliability Modeling of Resources

Indirect use of prioritization information can also be used to formulate new types of functionality for systems. For example, due to their collaborative and decentralized nature, federated grid and high-throughput computing systems are often unable to provide advance reservation and large-scale coallocation of resources. Incorporation of prioritization information in job distribution gateways could allow formulation of reservation

systems, where prioritization information is used in job queues to reserve resources for prioritized tasks.

For example, in decentralized desktop grid or volunteer computing systems constructed as distributed BOINC [2] networks, decentralized prioritization can be employed in computation gateways (BOINC servers) to provide dynamic prioritization of computational tasks. Formulated to provide differentiation in task-resource matching, such mechanisms could (by dynamically altering task prioritization when distributing tasks to clients) act as reservation systems that, e.g., reserve more reliable resources for higher prioritized tasks (a type of resource reservation where the likelihood of job completion within job deadlines is proportional to the priority of the job) or reserve resources with specific abilities (e.g., GPU computation capabilities or larger storage facilities) for high prioritized tasks at specified time slots.

Decentralized prioritization could also here be used to implement advanced non-linear scheduling criteria, e.g., partitioning of resource sets based on reliability modeling of resources. This could be used to for example enact optimized meta-scheduling resource plans for tasks in computational workflows on more reliable resource subsets, while simultaneously schedule independent bag-of-task computations on resources that in the past have behaved less predictably. Alternatively, decentralized prioritization could also be used to optimize computational throughput for volunteer computing environments by modeling the number of redundant computations scheduled versus some model of resource reliability (i.e. steering the amount of redundant computations scheduled on unreliable resource pools). For such modeling, different scheduling policies would be attributed different criteria in policy target trees. Resource ranking could for example be expressed by heuristics that model linear combinations of average resource capacity produced and the historical reliability of the resource (effectively expressing the expected value of capacity production for the resource).

F. Priority as a Currency

Prioritization information can also be used to allow end-users to (temporarily) influence the priority of their tasks in computational infrastructures. For example, it is conceivable to construct task brokering systems that allow end-users to request increased priority for a job at the expense of a higher metered cost for that job, or reduced priority for future jobs. Capacity metrics and allocation schemes here remain unaltered, but job submissions are attached meta-information defining requests for priority alterations, and capacity measurements are updated to reflect the altered priority of tasks in metering and accounting.

Similar use cases can also include to allow end-users to request reduced priority for jobs to be able to submit more jobs in the same quota (similar to increasing process nice values for background processes in operating system scheduling). Closely related scenarios could also be found in modification of quality-of-service related aspects of existing systems, e.g., high-performance computing scheduler backfilling that prioritizes important jobs, high-throughput computing scenarios where higher prioritized tasks are assigned to resources less likely to preempt computations, or commercial cloud computing scenarios where applications try to use spot prices (i.e. temporarily cheaper IaaS offerings during low load times) to instantiate virtual machines when possible.

In extension, use of priority as a currency could also allow formulation of economic models for bartering and trading of capacity. For example, capacity markets (similar in concept to commodity or stock markets) could be constructed to provide economic trading of economic derivatives for capacity production much like deregulated energy (electricity) markets. Auction-based models where capacity producers and consumers can trade futures (contracts to buy or sell specified amounts of capacity at specified times) and options (contracts for the right to buy or sell capacity at a specific price) can here be envisioned. For the decentralized prioritization model proposed in this work, prioritization of capacity requests could then be influenced by both prices set on capacity as well as bartering mechanisms that dynamically adjust the capacity consumption data used in priority calculation. Here prioritization-based management systems become not only controllers of other systems, but also producers of control data usable in a wide range of computation scenarios, e.g., agent-based simulation and trading systems.

G. Management of Management Systems

Decentralized prioritization information can also be used in control and tuning of infrastructure management systems. For example, control of heat production is central to minimization of energy costs for data centers. Decentralized prioritization systems can here be used to model optimal heat production scenarios (i.e. set target functions for heat production and distribution for separate parts of data centers), detect anomalies in system behavior (e.g., abnormally high or low load in individual servers or clusters), and autonomously steer local system management actions (e.g., instantiate or migrate virtual machines, wake or suspend servers). In such scenarios, the capacity modeled in policy target trees would not be the direct capacity produced (e.g., CPU cycles or storage capacity), but rather some form of capacity implied by resource behavior (e.g., power consumption or heat production), and system actions would be formulated in terms of high-level management system actions rather than requests for infrastructure capacity.

Management of management systems is typically more complex than steering capacity consumption directly, and is likely contain multiple conflicting goals and actions, e.g., computational infrastructures simultaneously attempting to maximize capacity production and minimize power consumption. Decentralized formulation of prioritization-based management systems will here include construction of more advanced (compound) metrics for system behavior, as well as heuristics for evaluation of system actions.

H. Decentralized Management and Control Theory

Decentralized prioritization can also advantageously be combined with classic control theory approaches. For example, conventional proportional-integral-derivative (PID) controllers are widely applicable and used in control systems. However, while formulation of PID controllers can very accurately capture and control system behavior, such systems do not encompass distribution of data and are generally sensitive to latencies in data updates. Combination of a system built on the proposed model for decentralized prioritization-based management and a conventional PID-based controller can be achieved e.g., by feeding priority vectors as input to PID controllers. In this case, the prioritization-based management framework would act as a decentralized data filter and distribution middleware for PID controllers.

Similarly, conventional control systems can also be used to steer (or modulate) the input data to prioritization-based management systems, to for example modulate what prioritization-based system to use based on some classification of the current situation, or to provide control for what information is disseminated to other nodes in decentralized systems. Another envisioned use for control theory-based controllers in the framework is to be employed as priority operators, potentially in combination with machine learning based approaches to aid in data structuring and filtering. Combination of control theory controllers and decentralized prioritization-based systems is expected to be a very expressive way to address construction of management systems and is subject for future work.

I. High-Level Priority-Based Management

For construction of more advanced prioritization mechanisms, prioritization-based management systems can also be cascaded and used in settings similar to filter banks. Consider for example the problem of Service-Level Agreement (SLA) management in IaaS cloud environments. It lies in the interest of the cloud provider to maximize the amount of customers using the infrastructure, e.g., by admitting more virtual machines than there are physical machines available (oversubscription). To minimize SLA violations (and avoid costly compensation penalties), infrastructure providers must ensure that end-users receive capacity commensurate to their subscriptions.

In this example, a cascade of prioritization-based management systems can be used to construct a system that automatically regulates capacity distribution according to policy target functions that can be intuitively formulated in terms of share allocations. For example, this could be formulated using three prioritizers. A round-robin type of prioritizer (that evenly awards all customers $1/n$ of the available capacity) could be used to ensure that a certain share (say 20%) of the capacity produced is reserved to meet minimal SLA levels. Additionally, a target function oriented prioritizer that distributes remaining capacity (80%) proportional to policy target allocations could be operating in parallel with this system. Finally, another prioritization-based manager could be sequentially coupled to these two to ensure that each manager's decisions is enacted the appropriate amount of times (20% and 80% respectively). In terms of system actions, this would mean that 20% of the request processing is made by the round-robin system, which ensures that 20% of the produced capacity is divided evenly among all customers. Note that this implies that even customers with policy target allocations lower than $0.2/n$ receive this minimal level.

Additionally, prioritization-based systems also capture semantics for recovery scenarios, e.g., in the example determine in what order to violate SLAs when it is unavoidable to do so (i.e. selecting SLAs to violate based on minimal cost or lowest priority). Naturally, prioritization-based management can also be combined with advanced automation tools such as machine learning and statistics tools, or classic control systems. In general, not all control problems can be posed as direct prioritization problems, but as discussed, it is often possible to extract prioritization components from control systems and treat them separately. The proposed model for decentralized prioritization is applicable for construction of generalized control frameworks, direct tools, and also for use as a decentralization mechanism in distributed scenarios.

Decentralized prioritization-based management systems also find applications in control of infrastructures at a higher level. For example, commercial cloud IaaS offerings provide the option of renting computational capacity instead of buying it, typically in the form of virtual machines instantiated through APIs and distributed interfaces. However, while hardware-enabled virtualization techniques effectively provide end-users the ability to instantiate virtual machines of different sizes and capabilities (vertical elasticity), as well as dynamically adjust the number of resources used (horizontal elasticity), cloud infrastructures still only implement partial aspects of the desired characteristics of cloud environments, and significant challenges remain to improve the flexibility and utility of cloud infrastructures [27], [28]. In particular, current IaaS offerings largely lack the flexibility and elasticity required to realize the vision of cloud computing (automated provisioning of ubiquitous and convenient access to shared pools of configurable computing resources) [30]. Limitations such as these hamper adoption of cloud computing, and act as bottlenecks in development of virtual infrastructures built on cloud resources [24]. Management systems constructed using decentralized prioritization can be used in cloud computing scenarios for a number of infrastructure control tasks, e.g., automated elasticity control (growing and shrinking, as well as duplication and removal of virtual resources), power management and energy efficiency (management of physical resources), and control of virtual machine live migration.

V. Conclusion

Computationally efficient decentralization models is likely to be an area of increasing importance as asynchronous parallelism is becoming more common in (heterogeneous and multi-core) hardware as well as in networks and storage systems. Prioritization is a core mechanic in many algorithms and an essential component in any economic system where demand exceeds supply (the case of most compute scenarios).

To facilitate decentralization of distributed management systems, we propose an approach to construction of management systems based on reformulation of control problems as prioritization problems, and present a model for decentralized, data-sharing management systems derived from a state-of-the-art fairshare scheduling technique. The approach emphasizes separation of data distribution from priority calculation and standalone prioritization algorithms that operate on distributed data sets. In the second part of the paper, we discuss application of decentralized management systems in distributed computing environments and outline key trade-off behaviors of the proposed model in selected application scenarios.

Preliminary results demonstrate the feasibility of the proposed model in large-scale distributed systems. For example, the Aequus system [21] demonstrates dynamic distributed quota enforcement on a scale applicable to grid and cloud computing scenarios. Decoupling task prioritization from task assignment in scheduling is illustrated in [31], which demonstrates performance improvements and synergistic effects from using prioritization information as feedback in high-level meta-scheduler systems. Results also indicate that properly formulated, decentralized prioritization mechanisms are capable of being scaled down and applied to more general tasks (e.g., chip-level prioritization of threads and processes), and that data partitioning schemes and distribution techniques can significantly reduce the I/O requirements of decentralized systems formulated as prioritization systems.

ACKNOWLEDGEMENTS

This work is supported by the High Performance Computing Center North (HPC2N) and funded by the Swedish Government's strategic research project eSSENCE and the Swedish Research Council (VR) under contract number C0590801 for the project Cloud Control. The authors acknowledge Gonzalo Rodrigo and Daniel Espling for work related to the project.

REFERENCES

[1] D. Abadi. Consistency tradeoffs in modern distributed database system design: Cap is only part of the story. *Computer*, 45(2):37–42, 2012.

[2] D. Anderson. BOINC: A system for public-resource computing and storage. In *5th IEEE/ACM International Workshop on Grid Computing*, pages 4–10, 2004.

[3] E. Brewer. Towards robust distributed systems. In *Proceedings of the Annual ACM Symposium on Principles of Distributed Computing*, volume 19, pages 7–10, 2000.

[4] E. Brewer. Pushing the cap: strategies for consistency and availability. *Computer*, 45(2):23–29, 2012.

[5] X. Calsamiglia and A. Kirman. A unique informationally efficient and decentralized mechanism with fair outcomes. *Econometrica: Journal of the Econometric Society*, pages 1147–1172, 1993.

[6] J. Celaya and L. Marchal. A Fair Decentralized Scheduler for Bag-of-Tasks Applications on Desktop Grids. In *Proceedings of CCGRID '10*, pages 538–541. IEEE Computer Society, 2010.

[7] E. Dafouli, P. Kokkinos, and E. Varvarigos. Fair Execution Time Estimation Scheduling in Computational Grids. In P. Kacsuk, R. Lovas, and Z. Nemeth, editors, *Distributed and Parallel Systems*, pages 93–104. Springer, 2008.

[8] J. De Jongh. *Share scheduling in distributed systems*. PhD thesis, Delft Technical University, 2002.

[9] E. Di Nitto, D. J. Dubois, and R. Mirandola. On exploiting decentralized bio-inspired self-organization algorithms to develop real systems. In *Software Engineering for Adaptive and Self-Managing Systems, 2009. SEAMS'09. ICSE Workshop on*, pages 68–75. IEEE, 2009.

[10] N. Doulamis, A. Doulamis, E. Varvarigos, and T. Varvarigou. Fair scheduling algorithms in grids. *Parallel and Distributed Systems, IEEE Transactions on*, 18(11):1630–1648, 2007.

[11] E. Elmroth and P. Gardfjäll. Design and evaluation of a decentralized system for Grid-wide fairshare scheduling. In H. Stockinger et al, editor, *Proceedings of e-Science 2005*, pages 221–229. IEEE CS Press, 2005.

[12] I. Foster. Globus online: Accelerating and democratizing science through cloud-based services. *Internet Computing, IEEE*, 15(3):70–73, 2011.

[13] D. Jackson, Q. Snell, and M. Clement. Core algorithms of the maui scheduler. In *Job Scheduling Strategies for Parallel Processing*, pages 87–102. Springer, 2001.

[14] J. Kay and P. Lauder. A fair share scheduler. *Communications of the ACM*, 31(1):44–55, 1988.

[15] K. Keahey. Nimbus: open source infrastructure-as-a-service cloud computing software. In *Workshop on adapting applications and computing services to multicore and virtualization, CERN, Switzerland*, 2009.

[16] J. O. Kephart and D. M. Chess. The Vision of Autonomic Computing. *Computer*, 36:41–50, 2003.

[17] K. H. Kim and R. Buyya. Policy-based Resource Allocation in Hierarchical Virtual Organizations for Global Grids. In *Proceedings of SBAC-PAD '06*, pages 36–46. IEEE Computer Society, 2006.

[18] K. H. Kim and R. Buyya. Fair resource sharing in hierarchical virtual organizations for global grids. In *Proceedings of GRID '07*, pages 50–57. IEEE Computer Society, 2007.

[19] S. Kleban and S. Clearwater. Fair Share on High Performance Computing Systems: What Does Fair Really Mean? In *Proceedings of CCGRID '03*, page 146. IEEE Computer Society, 2003.

[20] A. Lakshman and P. Malik. Cassandra: a decentralized structured storage system. *ACM SIGOPS Operating Systems Review*, 44(2):35–40, 2010.

[21] P.-O. Östberg, D. Espling, and E. Elmroth. Decentralized scalable fairshare scheduling. *Future Generation Computer Systems - The International Journal of Grid Computing and eScience*, 29:130–143, 2013.

[22] R. Pagliari, Y.-W. Hong, and A. Scaglione. Bio-inspired algorithms for decentralized round-robin and proportional fair scheduling. *Selected Areas in Communications, IEEE Journal on*, 28(4):564–575, 2010.

[23] J. Price and T. Javidi. Decentralized and fair rate control in a multisector cdma system. In *Wireless Communications and Networking Conference, 2004. WCNC. 2004 IEEE*, volume 4, pages 2189–2194. IEEE, 2004.

[24] L. Ramakrishnan, P. T. Zbiegel, S. Campbell, R. Bradshaw, R. S. Canon, S. Coghlan, I. Sakrejda, N. Desai, T. Declerck, and A. Liu. Magellan: experiences from a science cloud. In *Proceedings of the 2nd international workshop on Scientific cloud computing*, pages 49–58. ACM, 2011.

[25] K. Rzadca, D. Trystram, and A. Wierzbicki. Fair Game-Theoretic Resource Management in Dedicated Grids. In *Proceedings of CCGRID '07*, pages 343–350. IEEE Computer Society, 2007.

[26] N. Sandell Jr, P. Varaiya, M. Athans, and M. Safonov. Survey of decentralized control methods for large scale systems. *Automatic Control, IEEE Transactions on*, 23(2):108–128, 1978.

[27] L. Schubert and K. Jeffrey. Advances in clouds - research in future cloud computing. http://cordis.europa.eu/fp7/ict/ssai/docs/future-cc-2may-finalreport-experts.pdf, May 2013.

[28] L. Schubert, K. Jeffrey, and B. Neidecker-Lutz. The future of cloud computing - opportunities for european cloud computing beyond 2010. http://cordis.europa.eu/fp7/ict/ssai/docs/cloud-report-final.pdf, May 2013.

[29] D. D. Siljak. *Decentralized control of complex systems*. Courier Dover Publications, 2012.

[30] The US National Institute of Standards and Technology (NIST). Definition of cloud computing. http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf, May 2013.

[31] L. Tomas, P.-O. Östberg, B. Caminero, C. Carrion, and E. Elmroth. Addressing QoS in grids through a fairshare meta-scheduling in-advance architecture. In *Proceedings of 3PGCIC '12*, pages 226–233. IEEE, 2012.

[32] A. Yoo, M. Jette, and M. Grondona. SLURM: Simple Linux Utility for Resource Management. In D. Feitelson, L. Rudolph, and U. Schwiegelshohn, editors, *Job Scheduling Strategies for Parallel Processing*, volume 2862 of *Lecture Notes in Computer Science*, pages 44–60. Springer Berlin / Heidelberg, 2003.