Scheduling Policies for Enterprise Wide Peer-to-Peer Computing

Adnan Fida
Department of Computer Sciences
University of Saskatchewan
57 Campus Drive
Saskatoon, SK, Canada
+1 306 966-4886
adf428@mail.usask.ca

ABSTRACT
Peer-to-Peer (P2P) computing is a distributed computing paradigm which considers unreliable peer collections as target platform for application execution. In this paper such collections are considered in the context of enterprises. Unlike other P2P scenarios, more control and knowledge is available in regards to peers. This paper focuses on application task scheduling for such environments. Environment and application characteristics are identified and used to develop a set of characteristics for task scheduling policies. This characteristic set is then used to survey scheduling policies in similar distributed computing paradigms to find their relevance for enterprise P2P computing. A proof of concept simulation is presented for selected scheduling policies to outline some of the implementation decisions and validate the enterprise P2P computing paradigm.

General Terms
Algorithms, Design, Experimentation, Theory.

Keywords
Enterprise P2P Computing, Application Task Scheduling.

1. INTRODUCTION
The term Peer-to-Peer (P2P) computing has been around for at least seven years and has been defined and explored in many different ways. Typically any computing resource aggregation over a network which contributes towards a specific global task can be referred to as P2P computing. Generally, P2P computing is intended for data sharing, collaborative activities, and resource sharing such as CPU and disk space. There exists a wide range of variations among P2P computing environments in terms of size, scope, and characteristics [1]. For the purpose of this discussion, P2P computing is referred to as enterprise P2P computing, for reasons described below.

Currently enterprises such as businesses, educational institutes, and governmental organizations have collections of computing resources, such as desktops, which are often underutilized. For example, in a computing lab at a university, all desktops are not continuously being used and students login to desktops at random intervals and remain logged in for variable amount of times. Overall only a fraction of these desktops will be used and the idle desktops can be pooled. Each desktop (or other computing resources) in such pools can act as a peer. The peers collaborate with each other to achieve a common goal. Underutilized collections of computing resources present a viable opportunity for applications that can benefit from such an aggregation of peer computing power. Some examples of applications and areas which can benefit from enterprise P2P computing are simulations, financial risk modeling, and bioinformatics. Moozakis [2] describes some of the results from an enterprise P2P computing environment consisting of 100,000 desktop based peers. This environment allows complex and time consuming number crunching application execution for an aircraft engine modeling purposes. This rather extreme implementation of enterprise P2P computing has achieved 85% desktop processor utilization, which had been previously averaged at 5%.

However, in such computing environments, peer availability is unpredictable and therefore unreliable since local peer owners (students or employees) can take over the control of any resource at anytime.

In an enterprise information is either available or can be collected about peers, users, and their environment. The total number of peers that can be pooled together is known and can be considered as a fixed set such as P = *{p1, p2, p3, …, pn},* where *n* is total number of peers. For each peer *p in* set P, its configuration can be described as a tuple <*software, hardware*>, where *software* is a tuple of <*OS, applications, data*> and *hardware* is a tuple of <*memory, CPU, disk*>. Each peer *p* can be either *available* or *unavailable* at any moment in time (availability state). Overtime, this availability state can be determined for all peers in P. This availability model can then be used during application execution to compensate for the unreliable nature of peers. Other infrastructure information such as peer network topology, and peer access control are also known. Given this information, one or more applications can be developed and deployed in such computing environments. For an application set such as A = *{a1, a2, a3,…, am},* where *m* is total number of applications, each application can consist of more than one tasks. Every *a* in A can be specified as a set of tasks {*t1, t2, t3,…, tj}*, where *j* is total number of tasks for application.

Presence of this information makes enterprise P2P computing different from other types of P2P environments [1]. In addition, an enterprises owns all of the peers in set P= *{p1, p2, p3,…, pn}* and therefore, can provide guidelines for configuration, and operation of peers. Together this extends greater overall control to distributed applications set A = *{a1, a2, a3,…, am}* for their development, deployment and execution.

One of the requirements for application execution in such environments is to schedule all the of application tasks {*t1, t2, t3, …, tj}* across peers *{p1, p2, p3, …, pn}* with peers in an *available* state. This requirement can be further extended to simultaneously schedule tasks sets of more than one application in *{a1, a2, a3,…, am}*. Scheduling in distributed computing is a well studied area which has resulted in a diverse set of scheduling policies for various paradigms of distributed computing including Grid [3], Web-based computing [4], and domain specific heterogeneous distributed computing [5]. However, task scheduling for enterprise P2P computing is still an open issue. Due to the similarities between enterprise P2P computing and the above mentioned distributed computing paradigms, it is useful to examine existing scheduling policies in these domains to study scheduling policies for enterprise P2P computing. Since enterprise P2P computing is a fairly new concept with only a few (if any) real world deployments there is lack of available data to perform empirical studies. At this point, simulation based studies are the best available option.

In this paper a survey of existing scheduling policies for various distributed computing domains is presented. The remainder of this paper is organized as follows; section two, three, and four present characteristics of the enterprise P2P computing environment, applications and scheduling policies, section five, and six provide survey of existing scheduling taxonomies and policies, section seven presents proof of concept simulation, section eight concludes paper, and section nine contains references.

2. ENTERPRISE P2P ENVIRONMENT

In addition to the unreliable nature of enterprise P2P computing environments, there are other relevant issues that require awareness as well. Milojicic et al. [1] and Barkai et al. [10] name the following issues:

Heterogeneity; P2P computing environments are distributed in nature and they are generally formed of heterogeneous nodes/peers. The configuration peer tuples such as <*OS, applications, data*> and <*disk, CPU, memory*> often differ significantly even in an enterprise.

Autonomy; every peer is autonomous for its local users. That is, distributed application task will be superseded by tasks invoked by local user of peer. This autonomous nature of peers can lead to an *unavailable* state of the peer.

Scalability; in an enterprise P2P computing environment the size of pooled peer set P = *{p1, p2, p3, …, pn}* can be very large (*n* up-to thousands) and therefore, such environments require supporting mechanisms for distributed application execution. These mechanisms need to consider peer and task communication overhead, and some sort of load balancing across peers.

Security; although all of the peers belong to the same enterprise, due to local user autonomy, there are user concerns regarding what applications can access while executing on peers. For example, applications may need to ensure that they will not affect peer configuration for tuples <*OS, applications, data*> and <*disk, CPU, memory*>.

Performance & Cost; some performance and cost metrics are necessary for enterprise P2P computing environments in order to qualify them as viable alternative to equivalent centralized and expensive solutions.

State; in an enterprise P2P computing environment state information is available, since all of the peers in such environments are owned by the enterprise. As previously described state information can include peer availability at any instance in time, peer configuration and operational policies.

Other issues that arise due to unreliable nature of computing peers are:

Fault Resilience; mechanisms are required to handle faults generated by peers. Faults are generally generated when a peer changes is state to *unavailable* due to local user.

Transparency; fault resilient mechanisms are desired to take transparent actions for application tasks and should not require monitoring. However, in some cases it is possible to generate alerts to notify administrators when human assistance is required.

3. APPLICATIONS

Application development and deployment for enterprise P2P computing environments is rather challenging due to the unreliable nature of the peers. Applications not only have to exhibit some degree of parallelism, but also need to account for the notion that a task executing on a certain peer may fail or stop when a peer changes its state from *available* to *unavailable* due to interruptions by a local user. There are several characteristics for applications in such environments and some of the major ones are:

Parallelism; number of tasks in the application task set {*t1, t2, t3, …, tj}* that can be mapped to peers with an *available* state in *{p1, p2, p3, …, pn}* simultaneously, specify the degree of parallelism for each application. Parallel tasks do not depend on each other to begin execution however they may communicate with each other during their execution. Since task scheduling and communication overhead can be larger than task execution overhead for a large peer set, a relatively high degree of application task parallelism is desired for an application in an enterprise P2P computing environment.

Type; application task's can be compute-intensive, data-intensive or of both types. Compute-intensive tasks either run large complex computations or have large set of computations to process. Data-intensive tasks need to analyze large sets of data. Depending on type of applications task sets, different scheduling policies maybe required for each type.

Size; application size matters in enterprise P2P computing and can be measured as a size of application task set {*t1, t2, t3, …, tj}*, where *j* is total number of tasks for an application. Each task *t* itself has its own execution length as well that contributes to the overall execution length of an application. Generally, large size and highly parallel applications are favorable for enterprise P2P environments otherwise small size applications may suffer form the implementation overhead in such environments.

Task Dependencies; application tasks in set {*t1, t2, t3, …, tj}* may depend on each other for execution and these dependencies can range from fully independent tasks to highly dependent tasks. Applications whose tasks are highly dependent on each other for execution may have a low degree of parallelism and therefore, may not be able to take advantage of all available peers.

Data Availability; data-intensive tasks execution will be affected if their assigned peer does not have the data required for processing locally. For a large size compute-intensive application data communication overhead can be significant and therefore, task scheduling for such application types must consider data availability in its decision making.

Given the above mentioned characteristics applications with a high degree of parallelization and moderate data requirements are the most suitable candidates for enterprise P2P computing environments. Tasks for such applications can be scheduled (using some auxiliary policy) across all peers for execution. Peers can complete their assigned tasks during their idle time. The average time interval for which peers are available is an important factor when deciding on applications for such environments. If the average task completion time is larger than the average peer availability time, then the application completion time may rise to a point where it may not be feasible to schedule such an application. Large size applications with some degree of inter-task dependencies and/or data requirements may become suitable for enterprise P2P computing environments, if more sophisticated supporting mechanisms are available. For example, fault-tolerance mechanisms which are either able to retain task state and optionally able to migrate task to another available peer, can allow less interrupted overall application execution.

4. TASK SCHEDULING

Task scheduling for enterprise P2P computing is a two step process; in the first step the peer selection decision is made and in the second step the assigned peer decides the order in which the tasks are to be executed. Since the second step is similar to scheduling in non-distributed computing, it is not of interest for this discussion. Only the first step is considered for rest of this discussion. More formally, scheduling in an enterprise P2P computing is a problem of mapping a set of tasks *{t1, t2, t3, …, tm}* to a set of peers *{p1, p2, p3, … pn}* for some enterprise goals. Enterprise goals can include reduced application completion times, better resource utilization, and reduced hardware costs.

Given the enterprise P2P computing and application characteristics in the previous sections, task scheduling for unreliable peers becomes a challenging task. Available peers at scheduling time may not remain available for the length of the assigned task completion times and a task may have to be moved to other peers during execution. Without proper attention to task scheduling in enterprise P2P computing, the application completion may not even be possible. Some of the characteristics of task scheduling in enterprise P2P computing environment are:

Scheduler Organization; there are many ways on how scheduler itself can operate in such distributed environments. Some possible options can include dedicated peers devoted to scheduler execution, collections of related schedulers each with unique goals, and decentralized schedulers which may execute on every peer and communicate with others to make scheduling decisions.

Scheduling Policy; depending on the information available for scheduling, policies may compute schedules at the start of the application (static) with no revisions, improve statically computed schedules with information that becomes available at run-time (hybrid) or only compute schedules at run-time (dynamic).

State Estimation; the unreliable nature of peers and dynamic scheduling policies require methods for environment state estimation (run-time information) to make mapping decisions. These methods can be either predictive or non-predictive.

Rescheduling Approaches; improvements to computed schedules are generally desirable due to continuous changing peer states and approaches can be employed to initiate rescheduling. Some of the possible options include periodic or event driven rescheduling.

Task Dependencies; given the application's nature, tasks may depend (for data or coordination) on each other for their execution. Therefore, scheduling policies need to account for these dependencies to avoid deadlocks and to provide optimized schedules.

Scheduling Overhead; time and memory consumptions for computing and executing schedules adds to the overall overhead of application execution.

Fault Resilience; unreliable peers require scheduling policies to consider peer failure while making scheduling decisions. For example, a dynamic scheduling policy, when selecting a peer for task execution needs to ensure that target peer is still in an *available* state. Further, a policy may also be able to reschedule a task in case of peer failures.

Ease of Implementation; this includes scheduling policies that can be easily implemented without requiring too much overhead.

Adaptability for Application Nature; once an enterprise P2P computing environment is setup, it maybe required to serve as an execution platform for more than one application. Due to the diverse type and size of applications, the ability of a scheduling policy to adapt to every application can further improve overall peer utilization.

Scheduling policies either need to implement support mechanisms themselves or rely on other available services in environments for some of the above mentioned considerations. For example, a scheduler needs to determine a peer state (available/unavailable) before a task can be assigned to it. This can either be done by polling the peer or requesting a network service to determine peer state. Other supporting mechanisms can include task communication, task monitoring and migration, and environment state determination mechanisms.

Scheduling in distributed computing is a well studied area which has resulted in a diverse set of scheduling policies for various paradigms of distributed computing including Grid [3], Web-based computing [4], and domain specific heterogeneous distributed computing [5]. However, task scheduling for enterprise P2P computing is an open issue and requires more research. Due to similarities between enterprise P2P computing and above mentioned distributed computing paradigms, it is intriguing to review at existing work towards scheduling policies in these domains. This study attempts of find existing scheduling policies relevant for enterprise P2P computing. The study is conducted in two phases; in the first phase existing scheduling taxonomies are examined; and in second phase existing scheduling policies from various similar paradigms are investigated.

5. SCHEDULING TAXONOMIES

There have been numerous attempts to develop taxonomies for distributed computing scheduling policies in order to compare

various policies. These existing taxonomies are worth examining for enterprise P2P computing scheduling characteristics, as described in section four, due to the similarities of enterprise P2P and other distributed computing environments. Further, they also provide base for understanding and evaluation of scheduling policies described in section 6. In this section, four such taxonomies are presented[1]. In table 1, a map of enterprise P2P scheduling characteristics to described taxonomies is presented.

Figure 1 display a classic distributed computing scheduling taxonomy developed by Casvant et al. [9]. In addition to hierarchical characteristics of the taxonomy, there are some flat characteristics of taxonomies that can be applied to more than one branch of tree simultaneously. For succinct representation of a taxonomy the flat characteristics are omitted from the tree structure. These additional flat characteristics include *adaptive*, *load balancing*, *bidding*, *dynamic reassignment*, and *probabilistic*. *Adaptive* policies dynamically make adjustments to reflect changes in an environment. *Load balancing* is to increase overall throughput by equally distributing loads across the environment. *Bidding* refers to negotiation between peers generating tasks and those available for execution. *Dynamic re-assignment* is the task migration to a more appropriate peer and *probabilistic* policies randomly make scheduling decision from a subset of available options as opposed to spending time computing every possible option.

The above mentioned taxonomy is extended by Ekmecic et al. [6] and considers the degree of application parallelism (application mode) and peer models. This work describes distributed computing as a three step process of parallelism detection, parallelism characterization, and peer allocation for task scheduling. Further, distributed computing is classified in terms of four combinations of application execution modes and peer (or machine) models. These combinations are *Single Execution mode/Single machine Model (SESM), Single Execution mode/Multiple machine Models(SEMM), Multiple Execution modes/Single machine Model (MESM), and Multiple Execution modes/Multiple machine Models (MEMM)* are defined. Given these four combinations, an extension of the scheduling taxonomy [9] is developed. First two new branches of *competitive* and *non-competitive* are added under *non-cooperative* to indicate nature of interaction among scheduled tasks. Secondly, a new characteristic of *Load Sharing* is introduced in the flat set of characteristics, which refers to efforts towards utilizing underused resources. Additionally, a middle-ware taxonomy for application and peer interaction is also developed in terms of heterogeneity support (network level, operating system level, and programming language level), application development support for parallelism characterization specifications and task scheduling issues, and data access techniques. In table 1, the column titled 'Casvant & Ekmecic' contains enterprise P2P scheduling relevant characteristics of the above mentioned taxonomies. The extended taxonomy covers core characteristics of enterprise P2P scheduling; however, qualitative characteristics of *ease of implementation* and *application adaptability* are not present in this taxonomy.

In distributed computing, peer discovery/dissemination and state estimation are supporting mechanisms for task scheduling. Maheswaran et al. [8] combine such supporting mechanisms with task scheduling together to define Resource Management System (RMS). A taxonomy for RMS is developed in terms of its design issues for Grid like systems with intent to view RMS implementation impact for scalability and reliability of systems. Design objectives are used to categorize environments into compute-intensive, data-intensive, and service availability. A requirement model for RMS is established with Quality of Service (QoS) and data integrity as core requirements. QoS is defined for scheduling requests in regards to the impact on already scheduled tasks and an ability to guarantee expected level of service. Data integrity refers the to security of underlying peers. This requirement model is then used to develop a RMS taxonomy for design issues such as peer organization, resource (peer and data) models and scheduling characteristics. Since task scheduling is done for available resources, effort is spent on identifying considerations towards interaction with resources. The taxonomy accounts for scheduler organization (centralized, hierarchical, decentralized), methods for state estimation (predictive, non-predictive) to enable scheduling decisions, rescheduling approaches (periodic, event-driven) to cope with the dynamic nature of resources, and nature of scheduling policy (dynamic and static). Scheduling characteristics from this work, which are relevant to enterprise P2P scheduling, are under column titled 'Maheswaran' in table 1. Since the taxonomy is focused on the design of the scheduler it does not address characteristics of *task dependencies* as it is considered an application specific characteristic. Further, QoS does not include *scheduling overhead*.

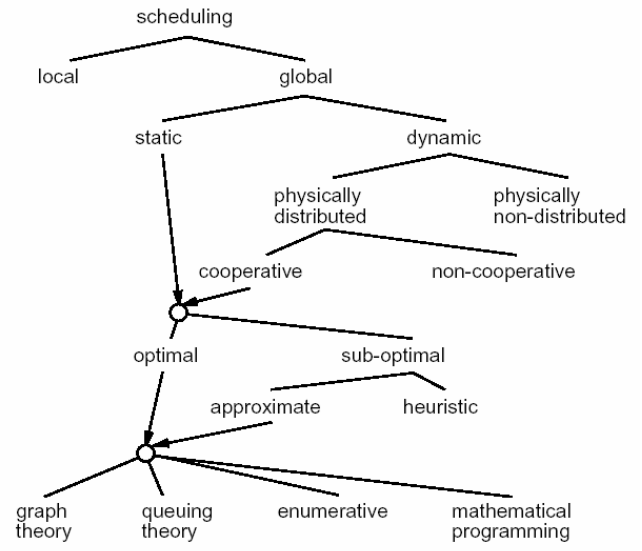

Figure 1. A Taxonomy of scheduling policies [9]

Braun et al. [7] provides a check list for the application and its execution environment characteristics, which can influence scheduling decisions. These characteristics are also used to extract scheduling policy characteristics and enterprise P2P relevant characteristics are in column titled 'Braun'. Some of the major

[1] A term of 'resource(s)' is used while describing these taxonomies and for rest of this discussion to refer to both peers and data, as both together requires consideration for task scheduling.

characteristics for application models include application size, its type (degree of task dependency), communication patterns, data availability, and QoS. For execution platform, characteristics include the number of available machines at any given time, number of network connections, machine architecture and degree of processing parallelism. Since this taxonomy considers application, platforms and scheduling, it is similar to scheduling characteristics of enterprise P2P computing.

All of the above mentioned taxonomies consider some or all of the enterprise P2P computing characteristics described in section 4, and therefore, can be applied when developing scheduling policies for enterprise P2P computing environments. However, the implementation of scheduling policies will vary from environment to environment and general purpose implementations will emerge overtime when existing limitations of enterprise P2P computing are removed. Some of these limitations described by Barkai et al. [10] include communication patterns and mechanisms, resource naming and discovery, peer availability, security and resource management. These limitations for now tend to favor applications with relatively independent tasks, requiring no or very limited communication, which can then be replicated across environment to deal with unreliable nature of peers.

6. EXISTING SCHEDULING POLICIES

In next three sub-sections existing scheduling policies are presented. The original set of enterprise P2P scheduling characteristics, presented in section four, is revised to exclude qualitative characteristics of *scheduling overhead, ease of implementation and adaptability*. Further, *re-scheduling approaches* are merged into *scheduling policy*. This revision is primarily done as it is hard to measure qualitative characteristics during empirical study and secondarily for reasons to stay succinct.

6.1 Grid

For Grid systems, a general purpose task scheduler is presented as a core tool for such an environment by Casanova et al. [11]. Component modularity is used in order to separate application implementation details from scheduling details. Applications express their performance requirements and data/task scheduling preferences for peers in order to benefit from this general purpose scheduler. Given these application requirements, the scheduler explores peers in the environment to determine the best possible schedules. Minimum turnaround time (scheduling time + application completion time) is used as criteria to decide on best schedules. Inclusion of multiple suitable peers and exclusion of peers that may not meet application specified requirements, are used as mechanisms to deal with the unreliable nature of peers. Search within a system for the best peers matching the application task requirement is the core of this scheduler design. In order to avoid exhaustive search across all peers in the system, various search heuristics are used. Schedules are only calculated once and the scheduler relies on other system implemented services (supporting mechanisms) or on users to gather information required to calculate optimal schedules. The scheduler's consideration to deal with the dynamic nature of peers makes it promising for enterprise P2P computing. It can also be extended to include failure rates as one of search heuristic while calculating optimal schedules. Given the overhead associated with gathering and comparing various schedules, applications with higher inter task dependency become suitable candidate for this scheduling approach.

Table 1. A map of enterprise P2P scheduling and existing scheduling taxonomies characteristics

Charct's.	Casvant & Ekmecic	Maheswaran	Braun
Organization	Distributed, Non-distributed	Centralized, Hierarchical, Decentralized	Control & Execution Location
Policy	Static, Dynamic	Fixed (sys./ app. oriented), Extensible (ad-hoc, structured)	Static, Dynamic
State Estimation	Approximate, Heuristic	Predictive, Non-predictive	Feedback
Rescheduling	Adaptive	Periodic, Event-Driven	Remapping
Task Dependencies	Cooperative, Non-cooperative	-	Depdencies, Task Duplication
Overhead	Optimal, Sub-optimal	-	Execution Times
Fault Resilience	Dynamic Reassignment	QoS (none, soft, hard)	Fault Tolerant, Task Duplication
Ease of Impl.	-	Resource Model	Feedback
Adaptability	-	Resource Discovery, Dissemination	App. Model supported

An interesting scheduling approach based on economic models is presented by Buyya et al. [12] for P2P Grid computing environment. It is assumed that in such environments end users drive task scheduling requirements for resources (peers and data). Users compete for these resources as these resource present value to users. For example, reporting results from scientific study early after aggregating more processors creates value of each processors for user. Task scheduling can be performed to optimize resource value to user, as opposed to typical some software or hardware optimization criteria (utilization, failure rates, efficiency). Since end user requirement driven resource management is also present in real world business economics, existing economic models are explored as one possible way to address task scheduling or resource management in distributed computing systems. Resources in such systems are viewed as services with service owner who can associate service access price for consumers. Therefore, access to these services per scheduling request can be negotiated (implemented) using economic models. Some of these models such as commodity market, posted price, bargaining, tender, auction, bidding, bartering, and monopoly/oligopoly are discussed for their implementation requirements. Systems are viewed to be composed of various supporting mechanisms, to facilitate service (resource) access in terms of its monitory values (price). Some of the examples for such supporting mechanisms include the Grid Trader Server, Grid Market Directory, Resource Brokers, Grid Info Service, and Grid Market Auctioneer. Further,

communication protocols and access mechanisms are also required as part of economic model implementations. In enterprise P2P computing, resources are owned by a single entity (i.e. enterprise) and therefore, the cost of resource consumption and benefits gained by tasks will balance each other.

Foster et al. [13] developed task scheduling policies for data Grids with the objective of reducing data access overhead, which is associated with remote calls by scheduled tasks. When tasks are scheduled at peers, which also host required data, significant improvements in overall application execution are achieved. Therefore, data Grid task scheduling policies are evaluated by accounting for data replication policies. In this work considerations are given to network bandwidth and latency, autonomous and unreliable nature of resources and size of system in terms of number of resources. There are two sets of scheduling policies; one for task scheduling and one for data replication. Together various combinations of these policies are studied to realize their impact on overall application execution. The first set includes JobRandom, JobLeastLoaded, JobRandLeastLoaded, JobDataPresent, and JobLocal, where as second set includes DataCaching, DataRandom, DataLeastLoaded, and DataRandLeastLoaded policies. These simple yet powerful policies appear promising for enterprise P2P computing for applications with data-intensive tasks, due to their low cost of implementation and decentralized nature. Implementation of these policies requires an external scheduler for mapping decisions and to deal with dynamic node nature.

6.2 Web-based Computing

In this section, systems with very large numbers (up-to millions) of computing peers communicating over web are considered. Mostly these systems have dedicated servers controlling the overall application execution.

Task scheduling for applications with a large number of compute-intensive large size tasks with high inter-task dependencies are studied by Rosenberg et al. [14]. Tasks and their dependencies are modeled as Directed Acyclic Graphs (DAGs) and in particular, scheduling policies for three common families of computation DAGs are developed. The primary goal of this work is to come up with scheduling policies that reduce the possibilities of computation deadlocks due to task dependencies on each other, and as a secondary goal require minimum memory. A web-oriented pebble game is used to model the execution of DAG nodes, where pebbles are placed or removed from peers according to dependencies described by DAG. These pebbles can then be of 'eligible' (eligible for scheduling) or 'executed' (finished execution) type. In order to achieve the before mentioned goals, the problem is to maximize the number of eligible pebbles at each step during scheduling. Scheduling is considered to be done on dedicated servers. For each DAG family, its structural properties are examined to determine with functions that maximize the number of eligible pebbles. This work focuses only on application deadlock elimination due to task dependencies and does not consider deadlock due to failure of task. Further, scheduling polices are only computed once. This work however is an important contribution towards scheduling highly inter-dependent tasks that are compute-intensive and can be applied to enterprise P2P computing if it can be extended further to consider rescheduling due to task failure (unreliable environment).

Table 2. Enterprise P2P scheduling characteristics in existing Grid scheduling policies

Charct's.	Casanova	Buyya	Foster
Organization	Centralized	Centralized (clustered)	Decentralized
Policy	Static	Static	Dynamic
State Estimation	Predictive	Predictive	Non-predictive
Task Dependencies	Dependent tasks on same peer	*Unclear*	Independent tasks
Fault Resilience	Duplication	Contract	Reactive

Mackie [15] describes a simple scheduling mechanism for a scientific study, in which large number of computation task are required to be executed in parallel. Like SETI@home, it relies on slave peers to poll masters for work. Every slave peer runs a scheduler task to determine when to poll master and when to perform execution when received with work. Upon local user interruptions, tasks are simply killed and execution is started again from the beginning when a peer becomes idle. No mechanisms to ensure application execution completions have been described. Since, applications with large but independent tasks are ideal for enterprise P2P computing, this type of scheduling can be very well adapted to enterprise P2P environments.

Foster et al. [16] presents a model to replicate data within networked resources to provide high data availability. A model is composed to determine the number of replicas necessary, and to determine when and where to replicate data. The number of replicas is determined by accounting for peer availability and accuracy of network state information of existing replicas. Periodic or request based approaches are used to trigger the replication decision making process, where as a cost based approach is used to determine where to replicate data. In an enterprise P2P computing system, the available system state information can be more accurate then in a large web-based system with unknown peers. Therefore, this model can be modified for state and replica information collection function to become more suitable for enterprise P2P computing.

The first ever work in looking into computing using peers available via the web is presented by Casanova et al. [17]. It considers SETI@home like applications which can be scheduled via a centralized scheduler onto heterogeneous peers (in terms of processing power, network bandwidth, and availability). Two simple algorithms for duplication and timeout are used. Either a task is duplicated enough to ensure task completion and have increase turnaround time at the cost of low throughput at the computing peer, or the timeout is set to avoid excessive duplication and waste of peer computing power. Simulations are performed by setting up a system model for various real world collected parameters and for various combinations of peer type percentages. There are two types of peers: aggressive (high availability) or conservative (low availability). It is found that on one extreme when excessive input tasks are available only timeout should be used to avoid low throughput, given quick turnaround time is not a concern, in the other extreme for short set

of input tasks, application duplication will provide better turnaround times. In general, for applications between these two extremes, turnaround time for the set of tasks should be use as a performance measure as opposed to single task turnaround time or peer throughput. Although, duplication and turn around based scheduling policies are possible within enterprise P2P computing, more sophisticated polices are attractive due to more state information availability.

6.3 Domain Specific Heterogeneous Distributed Computing

Scheduling policies described in this section are collected from domain specific heterogeneous distributed computing environments. These computing environments have custom implementations with various assumptions and limitations. Any scheduling work may not be directly applicable to enterprise P2P computing without reworking the supporting mechanisms for such policies. Nonetheless, these policies provide insight into issues encountered and possible implementation approaches for enterprise P2P computing.

A general purpose cost function is developed by Özgüner et al. [18], which can be incorporated in scheduling policies to include failure rate minimization as a criteria during scheduling decision making. This cost function is also independent of the underlying network topology and a static list heuristic based algorithm is extended to incorporate this cost function. Cost functions only considers network and hardware failure and do not address failures due to resource unavailability. Without an extension of this cost function (to consider unreliable nature of peers in enterprise P2P environment), it will not be able to provide any benefits since in enterprise, network and hardware failures can be considered to be relatively rare.

A hybrid scheduling policy is presented by Siegel et al. [19] which initializes with static algorithms and improves with information becoming available at run time. After the initial static schedule is computed, a two phase dynamic algorithm is applied. The first phase groups tasks into blocks such that no two tasks in same block depend on each other for data. These tasks are also ranked at this time with some predetermined scheme. The second phase has three variations and they all try to improve an initial statically computed schedule. The first variant tries to minimize tasks execution time of every block through remapping, the second variant reorders tasks within blocks every remapping step using task ranks and the third variant re-groups through a run-time computed parameter. This work focuses on minimum execution time for peers connected with high speed networks. Performance comparisons with other scheduling policies are performed and in some cases a 15% increase in performance is seen. These algorithms are shown to have better resource utilization due to overlapping of scheduling operations with task execution.

Simple dynamic algorithms are presented by Blake [20] for very constrained distributed computing environment, which does not take heterogeneity into consideration. Some of the environment simplifications include no task communication, task know their execution time, task migration consume equal processing time on target and originating resources, and task migration delays are independent of task size. Presented algorithms include No Scheduling (NS), Random Scheduling (RS), Arrival Balanced Scheduling (ABS), End Balanced Scheduling (EBS), and Continual Balanced Scheduling (CBS).Simulation studies reveal EBS to have better performance over CBS due to task migration overhead associated with CBS.

Table 3. Enterprise P2P scheduling characteristics in existing Web-based scheduling policies

Charct's.	Rosenbg	Mackie	Foster	Casanova
Org'n	Centld.	Centld.	Decentrald	Centld.
Policy	Static	Dynamic	Dynamic	Dynamic
State Est.	-	-	Non-predictive	-
Task Deped.	Deped. graph structure	Indeped. tasks	Indeped. tasks	Indeped. tasks
Fault Resilience	-	Task restart	Duplt	Duplt/ timeout

Kebbal et al. [21] implement a programming tool to provide means for parallel adaptive applications for heterogeneous distributed computing. This tool also includes a scheduling module, which is responsible for dynamically assigning application(s) tasks to available peers. Scheduling policy is responsible for partial or available DAGs mapping to peers in network using 'ready task' maximization heuristic. Machine load and task priority are used for mapping decision making. Computing environment is viewed to be of limited bandwidth and no local user autonomy is considered.

Atif et al. [22] develop a centralized dynamic algorithm, which is a modification of the branch-and-bound algorithm. It has considerably less complexity and performance is dramatically improved over the branch-and-bound algorithm. This is mainly due to the breaking of scheduling process into many phases, where during each phase only partial schedules are calculated. This algorithm also incorporates a cost model which then adapts to degree of heterogeneity in the system. Network bandwidth for communication overhead is given consideration, however, resource unavailability and other failures are not a concern. The developed algorithm is compared with other dynamic scheduling policies and algorithm is shown to have significant improvements.

Iverson et al. [23] give consideration to multiple applications (DAG in nature), which are competing for peers in single heterogeneous distributed environment. A scheduling framework is presented with algorithms for each component of framework. There are three decisions to be made in this framework: how to a make a scheduling decision, when to make a scheduling decision and when to place a task into a local peer queue. Functions are developed to capture the information required by these steps and all of this work is done dynamically. Computing environment is considered to be collection of heterogeneous resources spread over network, possibly around the globe. Further, it is also important to note that environment is not dedicated to one application and more than one application is competing for computing peers. The environment contains dedicated peers for running schedulers and it is realized that developed algorithms are

Table 4. Enterprise P2P scheduling characteristics in existing distributed computing scheduling policies

Charct's.	Özgüner	Siegel	Blake	Kebbal	Atif	Iverson	Jiang	Radulescu
Org'n	Centld.	Centld.	Centld.	Centld.	Centld.	Centld.	Centld.	Centld.
Policy	Static	Hybrid	Static/ Dynamic	Dynamic	Dynamic	Dynamic	Dynamic	Static/ Dynamic
State Est.	Non-predictive	-	Predictive	Predictive	Predictive	Predictive (heuristic)	Predictive (probabilistic)	Predictive
Task Deped.	Indeped. tasks	Indeped. tasks groups	Indeped. tasks	Indeped. tasks but prioritized	Indeped. tasks	Indeped. competing tasks	Increasing order of deadlines	Prioritized Tasks
Fault Resilience	Very Limited	-	-	-	-	-	Calculated assurances	-

practical and their simulations validate assumptions made for computing environment.

Jiang et al. [24] pay attention to scheduling issues in real-time distributed computing systems. Three dynamic heuristic based algorithms are presented; As Early As Possible (AEAP), As Late As Possible (ALAP), and Reliability Cost Drive (RCD). AEAP and AEAP do not take resource reliability into consideration where as RCD considers peer failure. All three algorithms also account for scheduling and dispatch time, which his critical in real-time systems. It is assumed that all applications have task execution times available in advance. Heterogeneous system with resources connected with high speed connections is viewed as execution platform with dedicated resources to perform scheduling.

A static FCP (Fast Critical Path) and dynamic FLB (Fast Load Balance) are presented by Radulescu et al. [25]. They try to reduce time complexity of task selection (FCP), and processor selection (FLB) policies. Both algorithms are modified to use task execution minimization as opposed to task start time minimization because former policy performs better in heterogeneous distributed computing. Simulations are performed for task sizes up-to 2000 and resources up-to 32. There is no consideration to peer failure, communication failure. FCP and FLB were compared with two other heterogeneous computing specific algorithms and performance results were acceptable in both cases. In case of irregular problems and high variance of processor speeds FCP and FLB degrade considerably.

7. SIMULATION

Enterprise P2P computing is relatively a new paradigm for distributed computing and there is very small number of real world deployments with mostly proprietary implementations. Due to this reason there is no data available to indicate peer availability models and application models. Without any real world data and access to a enterprise P2P computing environment, simulation is the only means of studying scheduling policies for such environments. In addition, simulators can also act as an incubator for various scheduling policies and for related supporting mechanisms before policies are applied to actual enterprise P2P computing environments. For the purpose of this discussion a proof of concept simulation is performed to validate the concept of enterprise P2P computing and to realize some of the implementation considerations for scheduling policies in such environments.

7.1 Scheduling Policies

Two scheduling policies of Random Assignment (RA) and Minimum Completion Time (MCT) are implemented in this simulation. RA is a static scheduling policy which randomly maps applications tasks to available peers upon their arrival. MCT is extension of RA, and after initial random mapping of tasks, MCT uses peer completion time to re-map tasks to peer with minimum completion time. Peer completion time is measured in terms of outstanding tasks in peer's local queue of assigned tasks. For every outstanding task, MCT obtains a peer with minimum completion time in the environment, and re-maps current task in case when initially assigned peer has longer completion time than newly obtained peer. Due to this ongoing remapping MCT becomes dynamic.

7.2 Environment

A simple enterprise P2P computing environment is modeled for this simulation, which has a centralized scheduler. Both scheduling policies and their supporting mechanisms are implemented on this centralized scheduler. RA and MCT both require a mechanism to obtain peers in the *available* state, whereas MCT also requires a mechanism to obtain a peer with minimum completion time at any instance in time. Both of these mechanisms are implemented by querying existing peers for their state and completion times. When more than one peer is possible the first peer in the result set is selected. Thus these two supporting mechanisms also provide state estimation for simulated environment. In addition on more explicit and predictive state estimation mechanism is also available to obtain the task execution status at any time. Tasks during their execution are simply terminated when the executing peer switch their state to *unavailable* and are restarted from the beginning when a peer becomes *available*. Given this heuristic no other fault tolerant mechanisms are present in the simulated environment. The peers generally tend to remain present in the environment and therefore, eventually all of the assigned tasks will be executed by assigned peers whenever peers are in *available* state. As an extension to the simulation model, task states can be retained to either continue task execution when peers go into *available* state, or to map task onto another

available peer. Rescheduling in this environment is only performed by MCT, which during its course of execution remaps tasks to peers with lower completion times. All of the application tasks are considered to be independent and non-co-operative and therefore all of the tasks can execute in parallel. However, in scenarios where tasks depend on each other for execution, some policies can be employed onto sub sets of tasks in which all of the tasks are independent of each other. Further, each of the tasks has same execution time which is known before hand. However, execution times are not used during scheduling decisions. Each of the peers is only capable of executing one task at a time and no peer level parallelization is modeled. The scheduling overhead for both RA and MCT is calculated in terms for elapsed time for scheduling policy completion. Other considerations for scheduling overhead can include communication overhead of scheduler and peers and CPU consumed towards executing a scheduling policy. Both of algorithms are relatively easy to implement and do not attempt to adapt to type and size of underlying applications.

7.3 Results

There are three variables selected for the simulation including peer set *{p1, p2, p3,..., pn}* (size of *n=10, 20, 30, 40, and 50*), task set *{t1, t2, t3, ..., tj}* (size of *j=50, 500*), and task length (*t=60, 3000, 6000, 9000*). The peer availability ceiling is set to twice as much as the task length and a random value is generated between 0 and availability ceiling for every peer at iteration of state change time. Two metrics of completion time and scheduling overhead are computed in terms of elapsed time units for combination of above mentioned variables. Some of the selected results are shown in figures 2, 3, 4, and 5. Since the primary intention of simulation is to realize enterprise P2P computing validity, a critique comparison between RA and MCT is not performed. MCT naturally by its description and collected results outperforms RA in every case, however MCT has a high scheduling overhead in every single case. Figure 2 and 3 are for a high load scenario with large number of tasks with longer execution times. In this case as the number of peers increase scheduling overhead of both RA and MCT tend to merge together, as more peers are available to MCT for load balancing and early completion of all tasks and hence MCT termination. Since, MCT runs as long as any tasks are outstanding, therefore, completion time and scheduling overhead in terms of elapsed time units for MCT are the same. Figure 4, and 5 present a scenario of small task set size with long execution time. In this case for large peer set sizes, RA completion times get close to MCT. Given the overhead attached with MCT, RA may become policy of choice for such scenarios.

From this simulation, enterprise P2P computing appears to be a valid heterogeneous distributed computing paradigm, which can increase enterprise resource utilization and provide alternative to expensive non-distributed solutions.

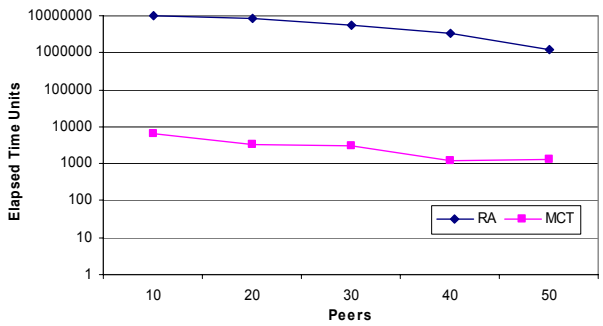

Figure 2. Completion time for 500 tasks of task length 9000

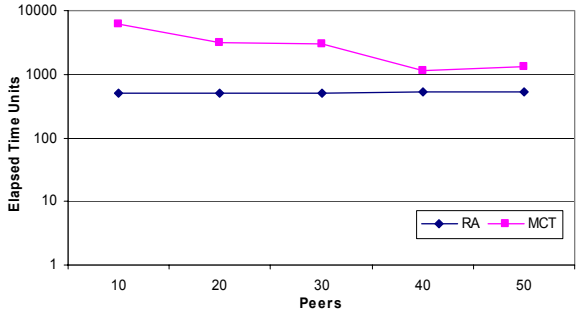

Figure 3. Scheduling overhead for 500 tasks of task length 9000

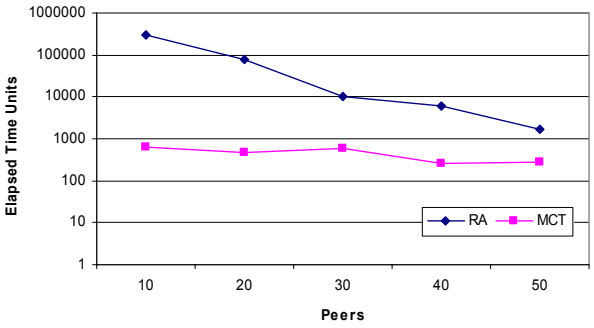

Figure 4. Completion time for 50 tasks of task length 6000

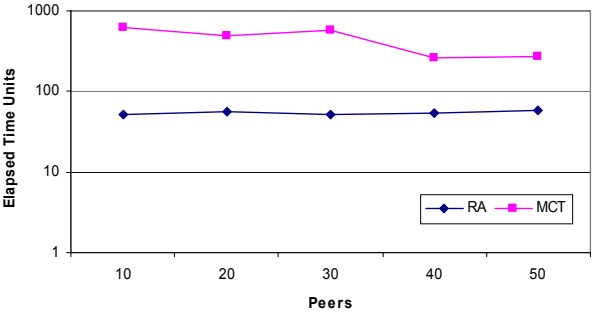

Figure 5. Scheduling overhead for 50 tasks of task length 6000

8. SUMMARY & FUTURE WORK

In this paper, enterprise P2P computing is introduced by identifying characteristics of its environment and suitable applications to study application task scheduling policies in such environments. Due to absence of task scheduling work and enterprise P2P computing similarities with other heterogeneous distributed computing paradigms, various scheduling taxonomies and policies from these paradigms are surveyed to identify their relevance for enterprise P2P computing. In addition, a proof of concept simulation is also performed to validate enterprise P2P computing concept. In a future more detailed simulation work is planned to study and develop sophisticated scheduling policies for enterprise P2P computing. Real world peer availability models and application models can be collected for purpose of this simulation with rather less restricted enterprise P2P computing environment.

9. REFERENCES

[1] Milojicic, D. S., Kalogeraki, V., Lukose, R., Nagaraja, K., Pruyne, J., Richard, B., Rollins, S., Xu, Z., Peer-to-Peer Computing. http://www.hpl.hp.com/techreports/2002/HPL-2002-57.pdf

[2] Moozakis, C., PCs As Supercomputer, http://www.internetweek.com/newslead01/lead052901.htm

[3] Foster, I., Kesselman, C., Tuecke, S. The Anatomy of the Grid: Enabling scalable virtual organizations. International J. Supercomputer Applications, 15(3), 2001.

[4] Milenkovic, M., Robinson, S. H., Knauerhase, R. C., Barkai, D., Garg, S., Tewari, V., Anderson, T. A., Bowman, M., Inter. Toward Internet Distributed Computing. Computer. Vol. 36(5), May 2003, pp. 38-46.

[5] Maheswaran, M., Braun, T. D., Siegel, H. J. Heterogeneous Distributed Computing. Encyclopedia of Electrical and Electronics Engineering, Vol. 8. J. G. Webster, ed., John Wiley, New York, NY, 1999, pp. 679-690

[6] Ekmecic, I., Tartalja, I., Milutinovic, V. A Survey of Heterogeneous Computing: Concepts and Systems. Proceeding of the IEEE, 84(8):1127-1144, Aug. 1996

[7] Braun, T. D., Siegel, H. J., Beck, N., Boloni, L., Maheswaran, M., Rerther, A. I., Robertson, J. P., Theys, M. D., Yao, B. A Taxonomy for Describing Matching and Scheduling Heuristics for Mixed-Machine Heterogeneous Computing Systems. Symposium on Reliable Distributed Systems. 1998: 330-335

[8] Krauter, K., Buyya, R., Maheswaran, M. A Taxonomy and Survey of Grid Resource Management Systems. Software Practices Experience 32(2): 135-164, 2002

[9] Casvant, T. L., Kuhl, J. G., A Taxonomy of Scheduling in General-Purpose Distributed Computing Systems. IEEE Transactions of Software Engineering, 14:141-154, 1988

[10] Barkai, D. Technologies for Sharing and Collaborating on the Net. First International Conference on Peer-to-Peer Computing, August 27 -29, 2001

[11] Dail, H., Casanova, H., Berman, F. A Modular Scheduling Approach for Grid Application Development Environments. Submitted to Journal of Parallel and Distributed Computing, April 8, 2002

[12] Buyya, R., Abramson, D., Giddy J., Stockinger, H. Economic models for resources management and scheduling n Grid computing. Concurrency and Computation: Practice and Experience 14(13-15): (2002)

[13] Ranganathan, K., Foster, I. Simulation Studies of Computation and Data Scheduling Algorithms for Data Grids. Journal of Grid Computing, V1(1) 2003

[14] Rosenberg, A., L., Yurkewych, M. Optimal Scheduling for Some Common Computation-Dags on the Internet. University of Massachusetts at Amherst, October 3, 2003

[15] Mackie, D., M. Simple and Effective Distributed Computing with a Scheduling Service. Army Research Laboratory, Adelphi, MD

[16] Ranganathan, K., Iamnitchi, A., Foster, I. Improving Data Availability through Dynamic Model-Driven Replication through Peer-to-Peer Communities. Global and Peer-to-Peer Computing on Large Scale Distributed Systems Workshop, Berlin, Germany, May 2002

[17] Kondo, D., Wing, E., Casanova, H., Berman, F. Models and Scheduling Mechanisms for Global Computing Applications. International Parallel and Distributed Processing Symposium, April 15-19, 2002

[18] Dogan, A., Özgüner, F. Matching and Scheduling Algorithms for Minimizing Execution Time and Failure Probability of Applications in Heterogeneous Computing. IEEE Trans. on Parallel and Dist. Systems 13(3): 308-323 (2002)

[19] Maheswaran, M., Ali, S., Siegel, H., J., Hensgen, D., Freund, R., F. A Dynamic Matching and Scheduling of a Class of Independent Tasks onto Heterogeneous Computing Systems. Proceeding of the 8th Heterogeneous Computing Workshop, April 1999

[20] Blake, B., A. Assignment of Independent Tasks to Minimize Completion Time. Software-Practices and Experience, Vol. 22(9), 723-734, September 1992

[21] Kebbal, D., Talbi, E., G., Geib, J., M. Building and Scheduling Parallel Adaptive Applications in Heterogeneous Environments. 1st IEEE Computer Society International Workshop on Cluster Computing, December 02-03, 1999

[22] Hamidzadeh, B., Lilja, D., J., Atif, Y. Dynamic Scheduling Techniques for Heterogeneous Computing Systems. Concurrency: Practice & Experiences, October 1995

[23] Iverson, M., A., Özgüner, F. Dynamic, Competitive Scheduling of Multiple DAGs in a Distributed Heterogeneous Environment. Heterogeneous Computing Workshop 1998: 70-78

[24] Qin, X., Jiang, H. Dynamic, Reliability-Driven Scheduling of Parallel Real-Time Jobs in Heterogeneous Systems, International Conference on Parallel Processing. September 03-07, 2001

[25] Radulescu, A., Gemund, Arjan J., C., van. Fast and Effective Task Scheduling in Heterogeneous Systems. Heterogeneous Computing Workshop 2000: 229-23