A Constraint Programming-Based Resource Management Technique for Processing MapReduce Jobs with SLAs on Clouds

Norman Lim and Shikharesh Majumdar
Dept. of Systems and Computer Engineering
Carleton University
Ottawa, ON, Canada
{nlim, majumdar}@sce.carleton.ca

Peter Ashwood-Smith
Huawei Technologies Canada
Kanata, ON, Canada
Peter.AshwoodSmith@huawei.com

Abstract—Clouds that are rapidly gaining in popularity require an effective resource manager that can harness the power of the underlying resource pool, and provide resources on demand to its users. This paper focuses on resource management on clouds for workflow requests characterized by Service Level Agreements (SLAs). Specifically, we devise a novel MapReduce constraint programming based resource manager (MRCP-RM) that can effectively perform matchmaking and scheduling of MapReduce jobs, each characterized by an SLA comprising an earliest start time, execution time, and an end-to-end deadline. Using discrete event simulation a performance evaluation of MRCP-RM is conducted for an open system subjected to a stream of job arrivals. The simulation results demonstrate the effectiveness of the resource manager and provide insights into system behaviour and performance.

Keywords—resource management on clouds, matchmaking and scheduling on clouds, SLAs on clouds, MapReduce with deadlines, MapReduce with SLAs.

I. INTRODUCTION

The popularity of cloud computing has been growing steadily over the past few years and it is being deployed extensively in the domain of Information Technology (IT). Well-known financial management and advisory companies, such as Merrill Lynch, and Gartner have predicted a multi-billion dollar market for the cloud computing industry [1]. The primary goal of cloud computing is to make hardware and software resources accessible as scalable, on-demand services over a network such as the Internet. More specifically, the cloud computing paradigm uses a service-oriented model that offers “everything-as-a-service” (XaaS). On such a system, a user can lease resources on-demand from a service provider and pay only for the time the resources are used. The “pay-as-you-go” model, scalability, and elasticity of the cloud that lets a user dynamically increase or shrink the number of resources allocated, are some of the attractive features of the cloud.

Cloud service providers typically deploy datacenters that house a large pool of resources including, computing, storage, and communication resources. Cloud computing that is based on resources acquired on demand is generating a great deal of interest among service providers and consumers as well as researchers and system builders. One of the key challenges in cloud computing is to devise effective resource management strategies that can harness the power of the underlying distributed hardware. Two of the essential functions that a cloud resource manager has to provide are matchmaking (or mapping) and scheduling. Matchmaking involves allocating resource(s) selected from a pool of resources to incoming requests. Scheduling determines the order in which the requests that are mapped on the same resource are executed. Matchmaking and scheduling are performed given certain user requirements and desired system objectives. User requests can be classified as on-demand requests that are to be executed on a best-effort basis, or requests that require quality of service (QoS) that is often captured in a service level agreement (SLA). When considering requests with SLAs, both matchmaking and scheduling are computationally hard problems because they need to satisfy the user’s QoS requirements while also considering system objectives such as maintaining high resource utilization to generate adequate revenue for the service provider.

Associating an earliest start time, execution time and deadline with a SLA characterizing a client request poses a new challenge to the resource management problem. Such requests are also referred to as advance reservation (AR) requests in the grid and cloud literature. Most of the research on resource management for ARs has only considered requests that require a single resource (i.e. single stage workflow). This paper concerns resource management on clouds for multi-stage workflow requests characterized by an end-to-end SLA. A multi-stage workflow request is a request that requires processing from multiple system components. An example of a multi-stage workflow is a MapReduce application (job). MapReduce is a programming model proposed by Google for processing large data sets (terabytes of data for example) [2], such as web request logs and other kinds of raw data, to generate more meaningful data sets. Since the input data sets can be very large, it is necessary to distribute the computation among multiple machines to facilitate parallel processing and reduce the total processing time. The MapReduce programming model has two key phases: a map phase and a reduce phase [2]. In the map phase, a map function is defined that accepts a set of input key/value pairs and generates a new set of intermediate key/value pairs. These intermediary key/value pairs are grouped together and then passed to the
reduce phase, where the intermediate key/value pairs are processed to generate a more meaningful data set.

MapReduce is often used in conjunction with cloud computing to help with Big Data analytics [3][4]. Many companies and institutions use MapReduce for a variety of different types of applications including: large scale data processing (e.g. sorting, indexing, and grouping), data mining (e.g. web crawling), artificial intelligence (machine learning), and scientific research (e.g. bioinformatics) [5]. For example, Google has used MapReduce applications to analyze web documents to generate search indices for its web search engine whereas Facebook uses MapReduce to analyze its user’s activities and the success of ads [6]. Thus, it is common for these companies and institutions to submit MapReduce jobs to a private cluster, or a cloud-based system. In both cases, a resource management middleware is required to handle the matchmaking and scheduling of the submitted MapReduce jobs on the underlying resources. MapReduce jobs with an associated completion time guarantee (or deadline) have also become important for latency-sensitive applications [7] such as those in the context of live business intelligence, personalized advertising, spam/fraud detection, and real-time analysis of event logs [8]. By allowing users to specify deadlines, the system can prioritize jobs and ensure that time-critical jobs are completed on time. Developing an efficient resource management middleware on such an environment is the focus of attention for this research performed in collaboration with Huawei, Canada.

More specifically, in this paper we focus on devising a resource manager that can effectively perform matchmaking and scheduling of an open stream of MapReduce jobs with SLAs comprising an earliest start time, execution time, and end-to-end deadline, on a cloud. Associating deadlines with MapReduce jobs has recently started to gain attention from researchers (e.g., see [7], [8], [9], and [10]). Considering earliest start times are important in the context of AR requests and are included in the SLAs considered in this paper. The matchmaking and scheduling problem for MapReduce jobs with SLAs is formulated using a constraint programming (CP) [11] methodology. Similar to linear programming (LP), CP is a theoretical technique used to solve optimization problems, and is capable of finding optimal solutions with regards to maximizing or minimizing an objective function [11]. However, unlike LP which has a theoretical basis formed on numerical algebra; CP is based on computer science principles, such as logic and graph theory [11]. In our preliminary work [12], we have compared a CP-based approach with a LP-based approach for resource management on a closed system running a fixed number of MapReduce jobs. The superiority of the CP-based approach, including its more intuitive and simple formulation of constraints, lower processing time overhead, and its ability to handle larger workloads, has motivated the investigation of the new CP-based MRCP-RM algorithm (presented in this paper) for handling open systems subjected to a stream of job arrivals.

CP has been shown to be effective in solving matchmaking and scheduling problems [13][14]; however, to the best of our knowledge, it has not been used for resource management in an open system subjected to a MapReduce workload. In this paper, a CP formulation that models the matchmaking and scheduling problem for MapReduce jobs with SLAs is devised. The CP formulation is implemented and solved using a Commercial-Off-The-Shelf software package with the objective of minimizing the number of jobs that miss their deadlines (i.e. late jobs). An advantage of using CP is that given a batch of jobs to map and schedule, an optimal mapping of tasks to resources and schedule of tasks executing on each resource that leads to the minimum number of late jobs can be generated.

Although significant work has been done on resource management for conventional jobs on clouds, less research exists in the area of matchmaking and scheduling MapReduce jobs with SLAs. This research is directed at filling this gap. To the best of our knowledge, there is no existing research on resource managers that use CP for matchmaking and scheduling MapReduce jobs with SLAs in a system subjected to an open stream of job arrivals, that this paper focuses on. The main contributions of this paper include:

- A CP formulation for matchmaking and scheduling MapReduce jobs with SLAs, and a discussion of using IBM ILOG CPLEX Optimization Studio [15] to implement the CP formulation.
- A new CP-based resource management algorithm and the MapReduce Constraint Programming based Resource Manager (MRCP-RM) for handling an open stream of MapReduce jobs. Addressing the challenges in devising and implementing the technique is discussed.
- A rigorous performance evaluation of MRCP-RM based on simulation using a variety of different system and workload parameters on an open system is performed. Insights gained into system behavior are described.
  - A novelty of MRCP-RM is captured in the low overhead CP-based technique that results in small values for the proportion of late jobs, ranging from 0.6% to 3.89%, for the open system and the workloads considered in this paper.
  - Performance analysis shows that MRCP-RM outperforms the resource manager described in the literature [8] by a substantial margin when the proportion of late jobs is the performance metric of interest.

The results of this research are useful to researchers, cloud providers as well as developers of resource management middleware for distributed systems, such as clouds.

The rest of the paper is organized as follows. Section II summarizes related work. Section III discusses our approach, including the problem model. Section IV focuses on the implementation of the CP formulation, and in Section V the design and implementation of MRCP-RM is presented. In Section VI, the results of the simulation-based experiments on MRCP-RM are presented and insights into system behavior are discussed. Lastly, Section VII concludes the paper and provides directions for future work.
II. RELATED WORK

A considerable body of knowledge exists in the area of resource management on grids and clouds for handling conventional sequential jobs [16]. One such example is the HEFT (Heterogeneous Earliest Finish Time) algorithm presented in [17]. HEFT is a task scheduling algorithm for heterogeneous computing environments that considers an application modelled as a directed acyclic graph (DAG) where the nodes represent tasks and the edges denote the precedence relationships. Similar to our work, the objective of HEFT is to simultaneously generate a high quality schedule and have fast scheduling time. However, there are some key differences between our work and HEFT. HEFT does not consider jobs (or applications) with an earliest start time (release time), or jobs with deadlines. In addition, HEFT is designed to consider only a single job (DAG) at a time, and does not specifically handle scheduling an open stream of job arrivals (multiple DAGs) simultaneously that our research focuses on.

The focus of this research is on MapReduce jobs characterized by multiple stages of execution, which have recently started receiving attention from researchers. As discussed in Section I, MapReduce is a popular application used by many organizations for Big Data analytics. A representative set of related works on resource management of MapReduce jobs is presented next.

In [18], the authors present an abstraction of the MapReduce matchmaking and scheduling problem by formulating it as an optimization problem using LP where the objective is to find a schedule that minimizes the completion time of the jobs. In [9], the authors describe cost-effective resource provisioning mechanisms for MapReduce applications with deadlines executing on the cloud. Dong et al. [10] focus on the scheduling of MapReduce workloads comprising of both jobs with and without deadlines. The proposed approach dynamically controls the execution of each job such that the jobs execute at their minimum degree of parallelism to meet its deadline. In [7], Mattess et al. describe a policy for dynamic provisioning of public cloud resources to schedule MapReduce jobs with deadlines. Initially, the jobs are executed on the local cluster, and if required resources from the cloud are dynamically provisioned to meet the application’s deadline.

In [8], Verma et al. propose a resource allocation policy based on earliest deadline first (EDF) called MinEDF-WC that allocates the minimum number of task slots required for completing a job before its deadline. MinEDF-WC can dynamically allocate and de-allocate resources (task slots) from active jobs as required. In our performance evaluation (see Section VI.B.1), we use a MapReduce workload derived from October 2009 Facebook workload traces that is also used in [8]. The workload traces were taken from a data-intensive system comprising a 600-node Hadoop warehouse that stores 2PB of data and runs about 7500 MapReduce jobs per day, processing 15TB of data.

The work presented in [18] uses a similar methodology, but the problem is formulated using LP instead of CP, and jobs with SLAs are not considered. The authors of [7], [8], [9], and [10], focus on developing heuristic-based schedulers that can handle jobs with deadlines. Also, the systems discussed in [7], [9], and [10] are not subject to an open stream of job arrivals, which MRCP-RM can effectively handle. Moreover, these papers do not consider SLAs comprising an earliest start time (e.g. AR requests) and deadlines for general MapReduce applications, which are handled by the techniques described in our paper. To the best of our knowledge, no existing paper has proposed a CP-based resource manager for handling MapReduce jobs with SLAs (comprising of an earliest start time, execution time, and an end-to-end deadline) in an open system with job arrivals that is described in this paper.

III. RESOURCE MANAGEMENT APPROACH

In Section III.A, the matchmaking and scheduling problem for MapReduce jobs with SLAs comprising an earliest start time, execution time, and deadline is presented. The CP formulation that is developed to solve the matchmaking and scheduling problem is also discussed in Section III.B.

A. Model of the Matchmaking and Scheduling Problem

The matchmaking and scheduling problem has two input components: a workload component, and a system component. The workload component describes the characteristics of the MapReduce jobs, whereas the system component defines the attributes of the resources that execute the jobs. The workload contains a set of MapReduce jobs to map and schedule: \( J = \{j_1, j_2, ..., j_n\} \) where \( n \) is the number of jobs in the set. Each job \( j \) in \( J \) has the following properties: (1) a set of map tasks \( T_m \) and reduce tasks \( T_r \) where \( k_m \) is the number of map tasks in job \( j \); (2) a set of reduce tasks \( T_r \) where \( k_r \) is the number of reduce tasks in job \( j \); (3) earliest start time for the job, \( s_j \); and (4) deadline for the job, \( d_j \) by which the job should be completed. Each map and reduce task \( t \) in \( T \) has an execution time (in seconds), \( e_t \), and a resource capacity requirement, \( q_t \). The execution times of the tasks include the time required to read the input data, and exchange data (e.g. intermediate keys) between the map and reduce phases. Note the value of \( q_t \) is typically set to one. A master set \( T \) contains the tasks for all the jobs in \( J \).

The jobs in the workload are to be handled one at a time. Each \( j \) is associated with a set of resources \( R = \{r_1, r_2, ..., r_m\} \) where \( m \) is the number of resources. Each resource \( r \) has a map task capacity (no. of map slots), \( c_r^m \), and a reduce task capacity (no. of reduce slots), \( c_r^r \). The map and reduce task capacity of each resource specifies the number of map tasks and reduce tasks, respectively, that the resource can run in parallel at a point in time. Note that investigating a model where \( c_r^m \) and \( c_r^r \) are not independent variables can form an interesting direction for future research.

The requirements for matchmaking and scheduling each job \( j \) in \( J \) on the set of resources \( R \) are summarized next. First, each job \( j \) can only be scheduled to start at or after \( s_j \). Each
A binary variable $x_{tr}$ that is used during matchmaking for mapping tasks to resources. If task $t$ is assigned to resource $r$, $x_{tr} = 1$, otherwise $x_{tr} = 0$. Each task $t$ in $T$ has an $x_{tr}$ variable for each resource $r$ in $R$.

- An integer variable $a$, that is used for scheduling of tasks on their assigned resources. Each task $t$ in $T$ has an $a$ variable that specifies the assigned start time of task $t$.

- A binary variable $N_j$ ($j$ in $J$) that indicates if a job $j$ misses its deadline (initialized to 0). If a job $j$ misses its deadline, $N_j$ is set to one; otherwise, $N_j$ remains zero.

A walkthrough of Table 1 is provided next. The objective function of the CP formulation states that the number of jobs that miss their deadlines (i.e. late jobs) should be minimized. The number of late jobs is calculated by summing $N_j$ for all jobs $j$ in $J$. Constraint (1) specifies that each task $t$ is assigned only to a single resource $r$. This is accomplished by iterating through all tasks in $T$ and ensuring that for each task $t$, the sum of all $t$’s matchmaking variables ($x_{tr}$) is equal to one. Constraint (2) iterates through the map tasks (specified in the set $T^{mp}_j$) of all jobs $j$ in $J$, and enforces that the assigned start time of the map tasks ($a$) is at or after the job’s earliest start time ($s$). Ensuring that each job’s reduce tasks are scheduled to start only after all of the job’s map tasks are completed is captured by Constraint (3). More specifically, Constraint (3) iterates through all of the jobs in $J$, focusing on the reduce tasks ($T^{rd}_j$) of each job $j$, and ensures that the scheduled start time of the reduce task ($a + e$) is greater than or equal to the completion time of the latest finishing map task (LFMT). Completion time of a task $t$ is equal to the scheduled start time of $t$ plus the execution time of $t$: $(a + e)$. The LFMT is found by passing the completion time of all map tasks of the job to the $\max$ function, which returns the maximum value in a set of values.

### Table 1. CP FORMULATION

<table>
<thead>
<tr>
<th>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimize $\sum_{j \in J} N_j$ such that</td>
</tr>
<tr>
<td>$\sum_{r \in R} x_{tr} = 1 \quad \forall t \in T$ (1)</td>
</tr>
<tr>
<td>$(a_t \geq s_j \quad \forall t \in T^{mp}_j) \quad \forall j \in J$ (2)</td>
</tr>
<tr>
<td>$(\max_{t \in T^{rd}_j} (a_t + e)) \quad \forall j \in J$ (3)</td>
</tr>
<tr>
<td>$(\max_{t \in T^{rd}_j} (a_t + e) \geq d_j \implies N_j = 1) \quad \forall j \in J$ (4)</td>
</tr>
<tr>
<td>$(\text{cumulative}(a_t</td>
</tr>
<tr>
<td>$(\text{cumulative}(a_t</td>
</tr>
<tr>
<td>$x_{tr} \in {0, 1} \quad \forall t \in T \forall r \in R$ (7)</td>
</tr>
<tr>
<td>$N_j \in {0, 1} \quad \forall j \in J$ (8)</td>
</tr>
<tr>
<td>$a_t \in \mathbb{Z} \quad \forall t \in T$ (9)</td>
</tr>
</tbody>
</table>

Constraint (4) states that if the latest finishing reduce task (LFRT) of a job $j$ completes after $j$’s deadline ($d_j$), then $N_j$ is set to one. Note that the LFRT is calculated in similar manner as the LFMT using the $\max$ function. Constraints (5) and (6) are the resource capacity constraints for map and reduce tasks, respectively. The CP global constraint, cumulative [20] is used to ensure that the capacity of each resource is not violated at any point in time. Specifically, at each point in time the cumulative constraint sums up the number of executing tasks, and enforces that this number is less than or equal to the resource capacity limit. The cumulative constraint requires four parameters including: the scheduled start time of the task, execution time of the tasks, resource requirement of the tasks, and the capacity for the resource. Each resource $r$ in $R$ has its own cumulative constraint, and only the tasks that are assigned to $r$ (i.e. tasks with $x_{tr}=1$) are of interest for the respective constraint. Finally, constraints (7) to (9) specify the domains of the decision variables, which are used to define the valid values that the respective decision variables can have.

### IV. Model Implementation and Solver

The CP formulation presented in Section III.B is implemented and solved using IBM ILOG CPLEX Optimization Studio v12.5 [15][21] (CPLEX for short). CPLEX’s integrated development environment (IDE) and Optimization Programming Language (OPL) [22] are used to implement the CP formulation. OPL is an algebraic language specifically designed for developing and expressing...
optimization models. The OPL model (implementation of CP formulation using OPL) is solved using CPLEX’s CP Optimizer constraint programming engine [21]. One of the advantages of modelling and solving matchmaking and scheduling problems using OPL and the CP Optimizer is that it does not require the enumeration (discretization) of time (i.e. specifying time periods are not required). The CP Optimizer also defines various specialized constraints and functions commonly present in scheduling problems, including precedence constraints, cumulative constraints, and control and synchronization constraints. For example, the CP Optimizer has a decision variable type called interval that can be used to represent tasks (or activities) that need to be mapped and scheduled. Section IV.A discusses how some of these features are used in the implementation of the CP formulation.

A. OPL Implementation of the CP Formulation

The implementation of the CP formulation (presented in Section III.B) using OPL, called the OPL model, is presented in this section. OPL [22] provides a data type called tuple, which is used to group together related data. The following tuples are defined: (1) Job = <id, earliest start time, deadline>, (2) Task = <id, parent job, type, execution time, resource capacity requirement>, (3) Resource = <id, map capacity, reduce capacity>, and (4) Alternative = <Task, Resource>. The parent job field in the Task tuple identifies which job the task belongs to. For example, if a task t has its parent job field set to 10, t belongs to the job with id equal to 10. The type field of the Task tuple is set to zero to indicate a map task, and set to one to specify a reduce task.

The inputs required by the OPL model include the following: a set of Job tuples named Jobs, a set of Task tuples named Tasks, and a set of Resource tuples named Resources. The Tasks set contains the tasks of all the jobs in Jobs. A set of Alternative tuples named Alternatives is then generated which contains all the possible Task and Resource combinations. The Alternatives set is used to represent the xᵦ decision variable discussed in Section III.B. The assigned start time, aᵦ, decision variable is implemented using an interval decision variable type as follows: dvar interval taskInterval [t in Tasks] size t.executeTime. This statement states that there is an array of intervals (one for each task t in the set Tasks), and each interval has a duration equal to the task’s execution time. Similarly, xᵦ is implemented using an interval decision variable called x: dvar interval x [a in Alternatives] optional. Note that this interval is set to be optional because there should only be a sub-set of x intervals present in the final solution (i.e., each task is assigned to one resource). Lastly, Nʲ is declared as a Boolean variable.

A representative set of examples of how the constraints of the CP formulation are implemented in OPL is provided. Constraint 1 which enforces that each task is assigned to one resource is implemented as follows:
for all (t in Tasks)
    alternative(taskInterval[t], all(a in Alternatives: a.task.id == t.id) x[a]);
The alternative function is used to ensure that only one of the x intervals is present in the final solution. The chosen x interval is selected from a subset of intervals that have the same id as the task of interest.

The resource capacity constraint for map tasks (Constraint 5) is implemented as follows:
forall (r in Resources) {
    sum (a in Alternatives: a.resource.id == r.id && a.task.isReduceTask == 0)
        pulse(alt[a].a.task.resReq) <= r.mapCapacity;
}
The pulse function is used to represent the resource usage of an interval, and is a function of time. When the task is active (executing), the pulse function increases the resource usage by the value qᵦ (set to one by default), and at other points in time the resource usage is zero. The functionality of CP’s cumulative constraint [20] is implemented by summing the values generated by the pulse function.

V. MapReduce Constraint Programming Based Resource Manager (MRCP-RM)

This section discusses the design and implementation of MRCP-RM, which is a CP-based resource manager that can perform matchmaking and scheduling of MapReduce jobs with SLAs on an open system. Without additional enhancements, the OPL model presented in Section III.B can only be used in a closed system where there is a fixed set of jobs known ahead of time. The focus of this paper is on open systems because real systems are often subject to an open stream of job arrivals.

A. Overview of MRCP-RM

Fig. 1 presents a diagram showing an overview of MRCP-RM. The resource manager, MRCP-RM, is implemented in Java (JDK7) using Netbeans IDE 7.3. The Job, Task, and Resource entities are abstracted and implemented as Java classes. The fields of the classes are similar to the tuple attributes described in Section IV.A with a few additions. The Job class has an arrival time field which stores the job’s arrival time. In the Task class, two additional attributes are present: isCompleted—set to true when the task is completed, and isPrevScheduled—set to true if the task is currently running on a resource and cannot be rescheduled.

In addition, the Java implementation of IBM’s ILOG Concert Technology API (Concert for short) and IBM ILOG OPL API [23] are used to embed the CP Optimizer and OPL models into the Java application. Section V.C discusses in more detail how MRCP-RM interfaces with IBM CPLEX.
As shown in Fig. 1, users submit MapReduce jobs to the system, which are then placed in a job queue. If MRCP-RM is not busy and there are jobs available in the job queue, MRCP-RM maps and schedules the jobs on to the resources (computing environment). This is accomplished by invoking the MRCP-RM algorithm (discussed in more detail in Section V.B), which incrementally builds on the previous solution (if one is available) by generating an OPL model with new constraints added for each of the tasks that have started but not completed executing. The new OPL model is then solved by CPLEX’s CP Optimizer to create an updated task to resource mapping and schedule for the system.

B. MRCP-RM Algorithm

The algorithm of MRCP-RM is presented in Table 2. After updating all of the earliest start times of the jobs (Lines 1-4), MRCP-RM checks all the scheduled tasks for each resource to see if the task has started executing (Lines 5-7). Tasks that have not started executing (i.e. earliest start time has not arrived) do not have to be checked further, whereas tasks that have started executing need to be further processed (Line 9). MRCP-RM stops checking the tasks of a resource once a scheduled task that has not started executing is found (Line 8). Note that MRCP-RM schedules all the newly submitted jobs (i.e. jobs in the job queue), but also remaps and reschedules the tasks that have not started executing to provide the most flexibility in scheduling the tasks and minimizing the number of late jobs. For example, a new job with an earlier deadline may need to be mapped and scheduled in the place of a previously scheduled job.

If the task has started executing (Line 9), MRCP-RM checks to see if it has completed its execution. If the task has finished executing, it is removed from J (Line 13-16). For each of the tasks that have started but not completed executing, MRCP-RM adds a new constraint (specifying the task’s assigned resource, start time, and end time) to the OPL model source code (Line 11). This constraint prevents the CP Optimizer from scheduling a new task on the same resource slot during the same time interval. For example, if the first task of job 3 (denoted t3_1) is scheduled to run on r_1 from 11 to 30 time units and has started running, the following constraint is added to the OPL model source code:

```c
forall (a in Alternatives : a.resource.id==1 && a.task.id=="t3_1")
  startOf(a)=11 && endOf(a)==30; }
```

In addition, the task’s `isPrevScheduled` attribute is set to true (Line 12) so that the solver does not apply Constraint 2 (enforcing earliest start times) to this task. The operations listed on Lines 19-24 are discussed in Section V.C.

C. Interfacing IBM CPLEX with MRCP-RM

MRCP-RM imports the following packages: `ilog.concert`, `ilog.cp`, and `ilog.opengl`, from the IBM CPLEX Java library. These APIs are used to implement the operations listed on Lines 19, 20 and 22. The implementation of each of these operations is briefly summarized next. More details on how to use these APIs can be found in [23]. To create the OPL model (Line 19), an instance of an `IloOplFactory` object is first created. The CPLEX Java library follows a Factory design pattern [23] where objects are created by invoking methods from a single master object. The OPL model is created by calling the `createOplModel()` method which requires passing in a model definition object, and a CP Optimizer instance. Generating the OPL model involves converting it to a Concert model, and is done by calling the OPL model’s `generate()` method. The OPL model is solved by calling the CP Optimizer’s `solve()` method (Line 20). After a solution is found, the values of the decision variables are extracted from the OPL model using Concert’s OPL element interface (Line 22).

<table>
<thead>
<tr>
<th>Table 2. The MRCP-RM Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input:</strong> A set of jobs J to map and schedule on a set of Resources R.</td>
</tr>
<tr>
<td>1: for each job j in J</td>
</tr>
<tr>
<td>2: if j’s earliest start time is less than the current time</td>
</tr>
<tr>
<td>3: Set earliest start time to current time</td>
</tr>
<tr>
<td>4: end for</td>
</tr>
<tr>
<td>5: for each resource r in R</td>
</tr>
<tr>
<td>6: for all scheduled tasks in r (tasks are sorted by start time)</td>
</tr>
<tr>
<td>7: if task’s start time is greater than current time</td>
</tr>
<tr>
<td>8: break</td>
</tr>
<tr>
<td>9: else</td>
</tr>
<tr>
<td>10: if task’s end time is greater than current time</td>
</tr>
<tr>
<td>11: Add a new constraint to the OPL model that specifies the start time, end time, and assigned resource of that task.</td>
</tr>
<tr>
<td>12: Set task’s <code>isPrevScheduled</code> field to true.</td>
</tr>
<tr>
<td>13: else</td>
</tr>
<tr>
<td>14: Record that the task is complete and remove it from the job’s tasks list.</td>
</tr>
<tr>
<td>15: if the job’s task list is empty</td>
</tr>
<tr>
<td>16: remove the job from J</td>
</tr>
<tr>
<td>17: end for</td>
</tr>
<tr>
<td>18: end for</td>
</tr>
<tr>
<td>19: Create a new OPL model and attach the data source containing J and R</td>
</tr>
<tr>
<td>20: Generate and solve the OPL model</td>
</tr>
<tr>
<td>21: if a solution is found</td>
</tr>
<tr>
<td>22: Extract and save the decision variable values (assigned start time and assigned resource) for each task.</td>
</tr>
<tr>
<td>23: else</td>
</tr>
<tr>
<td>24: Throw exception</td>
</tr>
</tbody>
</table>

D. Performance Optimization: Separating Matchmaking and Scheduling

Preliminary testing found that separating the matchmaking and scheduling operations in MRCP-RM can reduce the time required for the CP Optimizer to generate and solve the OPL model. The basic idea is to first perform the scheduling operation by solving an OPL model with the resource set R containing a single combined resource. This single combined resource contains the capacity of all the resources in the system. For example, consider a scenario where the system has 50 resources with `c_r^\text{mp} = 2` and `c_r^\text{rd} = 2`. In this scenario, the single resource `r` will have `c_r^\text{mp} = 100` and `c_r^\text{rd} = 100`. Using the CP Optimizer to solve the OPL model containing the single resource produces a single resource schedule, which contains the assigned start times of each of the tasks. Secondly, a matchmaking algorithm is used to map the tasks from the
single resource schedule to a new set of resources. In a batch workload with 25 jobs, and approximately 100 tasks in each job, it takes the system with one resource (100 map slots and 100 reduce slots) about 15 seconds to perform the matchmaking and scheduling operations. Conversely, on the system with 50 resources (two map slots and two reduce slots each), it took approximately 60 seconds.

The matchmaking algorithm has two input parameters: the number of resources that contain map slots (nm), and the number of resources that contain reduce slots (nr). The first step is to map the tasks from the single resource schedule to a set of resources that comprise of resources with \( c_r^{mp} = 1 \) and \( c_r^{rd} = 1 \) (called unit capacity resources). In the sample scenario, there will be 100 unit capacity resources. Each task \( t \) is mapped to the resource \( r \) that leaves the smallest remaining gap in the resource’s schedule. For example, consider a scenario in which there are two resources: \( r_1 \) and \( r_2 \), and a task \( t_1 \) that needs to be assigned to one of these resources from time 11 to 15. Resource \( r_1 \) has a task \( t_1 \) scheduled from time 2 to 10, and \( r_2 \) has a task \( t_2 \) scheduled from time 5 to 8. Task \( t_1 \) would be assigned to \( r_1 \) since the resulting gap will be 11-10=1 compared to the gap for \( r_2 \), 11-8=3.

The second step is to generate a schedule that contains the user-specified number of resources. The number of resources in the set is equal to \( \max\{nm, nr\} \), and the total number of task slots is divided evenly among the resources. In the sample scenario, if \( nm = 50 \) and \( nr = 30 \), there will be 50 resources, each with \( c_r^{mp} = 2 \), and 30 out of the 50 resources will have reduce slots. Specifically, 20 of the 30 resources will have \( c_r^{rd} = 3 \), and the remaining 10 resources will have \( c_r^{rd} = 4 \). Each user-specified resource is assigned a set of scheduled tasks from the unit capacity resources, the number of which is equal to its capacity. For instance, if a user-specified resource \( r \) has \( c_r^{mp} = 2 \), scheduled map tasks from two unit capacity resources are assigned to \( r \).

E. Performance Optimization: Earliest Start Time of Jobs

After performing preliminary experiments it was found that when there are many jobs that have arrived but have not started executing (i.e. their earliest start times, \( s_j \) have not arrived) the time required to perform the matchmaking and scheduling operations increases substantially. Recall that MRCP-RM maps and schedules new jobs as well as the tasks of previously scheduled jobs that have not started executing. In general, having more tasks to schedule increases the time required to generate and solve the OPL model because there are more decision variables to handle and more constraints to process. To reduce this overhead, a mechanism was implemented to start matchmaking and scheduling jobs only when their \( s_j \) have arrived, or are close to arriving. This reduces the number of tasks that MRCP-RM has to remap and reschedule each time it is run. Jobs that have arrived and have a \( s_j \) in the future are placed in a queue, and are mapped and scheduled at a later time.

VI. EXPERIMENTATION AND PERFORMANCE EVALUATION

In this section, a comparison of the performance of our proposed technique, MRCP-RM, with that of MinEDF-WC discussed in [8], is presented. This section also includes a detailed analysis of the impact of various system and workload parameters on system performance. Simulation is a popular method for performance evaluation of resource management techniques (see [8], [9], and [18] for example). In line with these works, a simulation-based approach has been used in this research to conduct the performance evaluation of MRCP-RM for an open system. A simulated system and workload provide the flexibility to systematically vary the system and workload parameters. Our performance evaluation does not concern obtaining exact values of the performance metrics achieved on a given system. Instead, we focus on the relative performance of MRCP-RM in comparison to the resource management algorithm presented in [8], and understanding the performance trends as captured in the degree of change in performance metrics in response to changes in workload parameters. Thus, a simulation-based approach is apt for answering the research questions addressed in this paper. For each simulation run, we consider the following performance metrics:

- Average matchmaking and scheduling time of a job (\( O \)), which is calculated as the total time required to perform matchmaking and scheduling of jobs during a simulation run divided by the total number of jobs mapped and scheduled. Note that \( O \) is a measure of MRCP-RM’s processing overhead.
- Number of jobs that miss their deadlines (\( N \)).
- Average job turnaround time (\( T \)), which is calculated as \( \frac{\sum_{j=0}^{N} (CT_j - s_j)}{N} \) divided by the number of jobs mapped and scheduled, where \( CT_j \) is the completion time of job \( j \).

Note that \( O \) is measured using Java’s `System.nanoTime()` method whereas \( N \) and \( T \) are produced as output by the CP Optimizer solving engine. We also define the percentage of late jobs, \( P \), as the ratio of \( N \) and the total number of jobs that arrived on the system. Recall from Fig. 1 that MRCP-RM runs on its own CPU, and the tasks of the jobs are executed on set of \( m \) resources, \( R \).

Table 3 outlines the workload and system parameters used in the experiments. The first row defines the number of map tasks (\( k_{map} \)) and reduce tasks (\( k_{rd} \)) for each job \( j \), whereas the second row specifies the execution time of the map tasks (\( me \)), and reduce tasks (\( re \)). Note that each job in the workload contains different values for \( k_{map} \) and \( k_{rd} \), which are both generated using discrete uniform (DU) distributions. Furthermore, \( me \) is generated using a DU distribution where \( e_{max} \) is the upper bound of \( me \), and \( re \) is generated based on the total \( me \) of the job as shown in Table 3. Thus, each task in a job has different execution times. As mentioned in Section III.A, the execution time of the tasks includes the time required to read the input data, and exchange data (e.g. intermediate keys) between the map and reduce phases.

The third row defines the earliest start times of the jobs (\( s_j \)), which can be equal to the job’s arrival time (\( v_j \)) or a time in the future after its arrival. The probability that \( s_j \) that is not equal to \( v_j \) is denoted by \( p \). Other works (see [8] and [9] for example) did not consider \( s_j \) as a workload parameter (i.e. for all jobs \( s_j = v_j \)). This paper investigates both jobs with \( s_j \) equal to...
$v_j$ and $s_j$ greater than $v_p$ and studies the impact of changing $p$ on system performance (see Fig. 6). The parameter $s_{\text{max}}$ is the upper bound of the value added to $v_j$ for generating its $s_j$ when $p$ is greater than zero. In the “Deadline of Job” row, $TE$ is the minimum execution time of the job assuming there are no other jobs in the system. The deadline of the job ($d_j$) is generated as the sum of $s_j$ and $TE$, multiplied by the value of the deadline multiplier, which is generated using a uniform (U) distribution where the parameter $d_{\text{UL}}$ is the upper bound of the distribution. The next row specifies the arrival rate of jobs ($\lambda$) which is produced using a Poisson process. Lastly, the final row defines the number of resources ($m$) in the system, as well as the map and reduce task capacities of each resource ($c_{\text{mp}}$ and $c_{\text{rd}}$, respectively).

An investigation of how changing these different experimental parameters can affect the performance metrics is conducted by performing factor-at-a-time experiments, where one parameter is varied and the other parameters are kept at their default values (shown in boldface in the ‘Values’ column) of Table 3. The different workload parameters are varied to generate workloads with different characteristics. The values and type of distributions used for the workload parameters: $k_j^{\text{mp}}, k_j^{\text{rd}}, me$ and $re$ were adopted from [18]. The parameters $s_j$ and $d_j$ (that were not used in [18]) are generated using uniform distributions similar to the other parameters. The use of a Poisson arrival stream for job arrivals is in line with [8].

<table>
<thead>
<tr>
<th>Name</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Workload Parameters</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Job:</strong></td>
<td></td>
</tr>
<tr>
<td>No. of map tasks, $k_j^{\text{mp}}$</td>
<td>$k_j^{\text{mp}} \sim \text{DU}[1,100]$</td>
</tr>
<tr>
<td>No. of reduce tasks, $k_j^{\text{rd}}$</td>
<td>$k_j^{\text{rd}} \sim \text{DU}[1,100]$</td>
</tr>
<tr>
<td><strong>Task Execution Time (in s)</strong></td>
<td></td>
</tr>
<tr>
<td>- Map task (me)</td>
<td>$me \sim \text{DU}[1, e_{\text{max}}]$ where $e_{\text{max}} = {10,50,100}$ $re = (3 \times \sum_{i=1}^{me} e_i)/k_j^{\text{rd}} + \text{DU}[1,10]$</td>
</tr>
<tr>
<td>- Reduce task (re)</td>
<td></td>
</tr>
<tr>
<td><strong>Earliest Start Time of Job, $s_j$ (in s)</strong></td>
<td></td>
</tr>
<tr>
<td>$s_j = \begin{cases} v_j, &amp; x = 0 \ v_j + \text{DU}[1,s_{\text{max}}], &amp; x = 1 \end{cases}$ where $v_j$ is the arrival time of job $j$; $x \sim \text{Bernoulli}(p), p = {0.1,0.5,0.9};$ $s_{\text{max}} = {10000,50000,250000}$</td>
<td></td>
</tr>
<tr>
<td><strong>Deadline of Job, $d_j$ (in s)</strong></td>
<td>$d_j = s_j + TE*U[1,d_{\text{UL}}]$ where $d_{\text{UL}} = {2,5,10}$</td>
</tr>
<tr>
<td><strong>Arrival rate of Jobs, $\lambda$ (jobs/s)</strong></td>
<td>$\lambda = {0.001, 0.01, 0.015, 0.02}$ (Inter-arrivals times-Exponential Distribution)</td>
</tr>
<tr>
<td><strong>System Parameters</strong></td>
<td></td>
</tr>
<tr>
<td>No. of Resources, $m$</td>
<td>$m = {25,50,100}$</td>
</tr>
<tr>
<td>- Map task capacity, $c_{\text{mp}}$</td>
<td>$m = {25,50,100}$</td>
</tr>
<tr>
<td>- Reduce task capacity, $c_{\text{rd}}$</td>
<td>$m = {25,50,100}$</td>
</tr>
<tr>
<td>Note: DU = discrete uniform distribution, U= uniform distribution</td>
<td></td>
</tr>
</tbody>
</table>

**Experimental Setup**

The experiments were performed on a PC with an Intel Core i5-4670 CPU (3.40GHz) and 16GB of RAM running Windows 8 Professional 64-bit. During each simulation experiment, MRCP-RM and the CPLEX solver were both executed on this PC. Each factor-at-a-time experiment was run long enough to ensure that the system operates at steady state. Each experiment was repeated a sufficient number of times such that the confidence interval for $T$ remains less than ±1% of the average value, at a confidence level of 95%. For $S$, the intervals were observed to be less than ±5% of the average value for all experiments except two cases were the value was less than ±7%. This resulted in a reasonable time for running the simulation experiments. The resulting accuracy of the simulation results is deemed adequate for the nature of the investigation presented in this paper: the focus of which is investigating the trend in the variation of a given performance metric in response to changes in the system and workload parameters. In the graphs presented in Fig. 4 to Fig. 9, $O$ and $T$ are displayed as a line graph and a bar graph, respectively, with the Y-axis in each graph having its own scale. Furthermore, the confidence intervals are shown as bars originating from the average value.

**B. Experimental Results**

MRCP-RM was configured to use three job ordering strategies, which determines the job MRCP-RM attempts to map and schedule first in its set of jobs to execute: (1) job id (jobs ordered by job id), (2) earliest deadline first (EDF), and (3) least laxity first (laxity of a job $j$, $L_j = d_j - s_j - \sum_{i \in j} e_i$). For maintaining clarity of the figures, only the results for EDF that produced the smallest $P_2$ are displayed. Using the other strategies did not produce any significant difference in the performance metrics.

1) Comparison with Related Work

To evaluate the effectiveness of MRCP-RM, it was compared to MinEDF-WC [8], a technique that we determined to be most closely related to our work: matchmaking and scheduling MapReduce jobs with end-to-end deadlines in an open-system (subject to job arrivals). The experiments described in this section were performed with a synthetic Facebook workload generated from October 2009 workload traces that is also used by [8]. More specifically, the workload consists of 1000 jobs where each job has a different number of map and reduce tasks as outlined in Table 4.

<table>
<thead>
<tr>
<th>Job Type</th>
<th>$k_j^{\text{mp}}$</th>
<th>$k_j^{\text{rd}}$</th>
<th>No. of Jobs</th>
<th>Job Type</th>
<th>$k_j^{\text{mp}}$</th>
<th>$k_j^{\text{rd}}$</th>
<th>No. of Jobs</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>380</td>
<td>6</td>
<td>200</td>
<td>50</td>
<td>60</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>0</td>
<td>160</td>
<td>7</td>
<td>400</td>
<td>0</td>
<td>40</td>
</tr>
<tr>
<td>3</td>
<td>10</td>
<td>3</td>
<td>140</td>
<td>8</td>
<td>800</td>
<td>180</td>
<td>40</td>
</tr>
<tr>
<td>4</td>
<td>50</td>
<td>0</td>
<td>80</td>
<td>9</td>
<td>2400</td>
<td>360</td>
<td>20</td>
</tr>
<tr>
<td>5</td>
<td>100</td>
<td>0</td>
<td>60</td>
<td>10</td>
<td>4800</td>
<td>0</td>
<td>20</td>
</tr>
</tbody>
</table>

By analyzing a graph of the cumulative distribution function (CDF) for the map and reduce task execution times, [8] determined that the execution times followed LogNormal distributions, $LN(\mu, \sigma^2)$ where $\mu$ is the mean and $\sigma^2$ is the variance. More specifically, the execution times for map tasks and reduces tasks (in milliseconds) are generated using $LN(9.9511,1.6764)$, and $LN(12.375,1.6262)$, respectively. Our own analysis of the CDF also confirmed these values. The
earliest start time of a job \( (s_j) \) is equal to the arrival time of the job (i.e. \( p=0 \)), and the deadline of the job \( (d_j) \) is the same as described in Table 1 where \( d_j \) is equal to two. As described in [8], the system used to map and schedule this workload comprises of 64 resources where each resource had one map slot and one reduce slot.

Fig. 2 and Fig. 3 show a comparison of \( P \) and \( T \), respectively, achieved by MRCP-RM and MinEDF-WC[8], when matchmaking and scheduling the jobs of the synthetic Facebook workload. The results presented are averaged over 100 simulation runs. The systems were subject to Poisson job arrivals, and the arrival rates that were chosen for comparison ensured that the simulation was stable. The results show that MRCP-RM leads to a significantly lower \( P \) in comparison to MinEDF-WC (see Fig. 2). The reduction in \( P \) is observed to change from 93% to 70% as \( \lambda \) is changed from 0.0001 to 0.0005 jobs/s. In addition, Fig. 3 shows that MRCP-RM achieves up to a 7% lower \( T \) in comparison to MinEDF-WC.

![Fig. 2. MRCP-RM vs. MinEDF-WC: Comparison of proportion of jobs that miss their deadlines.](image)

![Fig. 3. MRCP-RM vs. MinEDF-WC: Comparison of average job turnaround time.](image)

The following sections focus on analyzing the effect of various system and workload parameters on the performance of MRCP-RM.

2) Effect of Task Execution Time
The results presented in Fig. 4 demonstrate that \( O \) and \( T \) increase with \( e_{\text{max}} \). The longer mean task execution time increases the times required for jobs to complete, which increases \( T \) and causes tasks to remain in the system for longer periods. Recall that MRCP-RM creates and solves a new OPL model when jobs arrive, and adds a new constraint to the OPL model for each task that has started but not completed executing (refer to Section V.A). In general, adding more constraints increases the model generation and solve times, which is why \( O \) increases. However, as captured in Fig. 4, \( O \) is much smaller than \( T \): \( O/T \) was observed to remain lower than 0.02% when \( e_{\text{max}} \) is changed from 10s to 100s.

![Fig. 4. Effect of task execution times on system performance.](image)

3) Effect of Earliest Start Time
Fig. 5 shows that \( O \) and \( T \) tend to decrease as \( s_{\text{max}} \) increases. At higher \( s_{\text{max}} \), the execution of different jobs do not overlap as often: some jobs have earliest start times closer to their arrival times whereas other jobs have earliest start times further ahead in the future. This decreases the number of jobs that MRCP-RM has to map and schedule each time it is invoked, leading to a reduction in \( O \). In addition, \( T \) and \( P \) also decrease as \( s_{\text{max}} \) increases because it is not as likely that the execution of the jobs overlap, and thus the jobs can be scheduled to start executing at different points in time. Fig. 6, in which \( p \) is varied, shows a similar trend in performance as observed in Fig. 5. However, it is observed that the decrease in \( O \) is not as substantial as in Fig. 5 because the \( s_{\text{max}} \) values are not as high as those used in the experiments for Fig. 5.

![Fig. 5. Effect of earliest start time of jobs on system performance.](image)

4) Effect of Deadline
As shown in Fig. 7, as \( d_{\text{M}} \) increases, \( O \) decreases. When \( d_{\text{M}}=2 \), \( O \) is much higher compared to when \( d_{\text{M}} \) is 5 and 10. This is because when \( d_{\text{M}}=2 \), jobs have less laxity (slack time) and thus MRCP-RM requires more time to map and schedule the tasks in order to minimize the number of late jobs. It is observed
that $T$ is not significantly affected when $d_{ed}$ is increased. This is because only a small proportion of jobs has to be delayed to minimize $N$. A more significant change in $T$ would be observed if $\lambda$ is increased, and more jobs are present in the system at a given point in time. As expected, as $d_{ed}$ increases, $P$ decreases; $P$ is observed to be 3.46%, 0.56%, and 0.21%, when $d_{ed}$ is equal to 2, 5, and 10, respectively.

5) **Effect of Arrival Rate**

As expected, $O$ and $T$ increase with $\lambda$ (see Fig. 8). The main reason that causes an increase in $O$ is having many tasks that have started but not finished executing. This is because additional constraints need to be added for these tasks, which increases the model generation and solving times. A higher $\lambda$ causes this scenario to occur more frequently. This in turn increases the number of tasks that MRCP-RM has to map and schedule each time it is executed. As a result, it takes longer to generate and solve the OPL model, which leads to an increase in $O$. However, $O$ is still very small compared to $T$ even at higher $\lambda$ values. For example, $O/T$ was observed to vary from 0.005% to 0.04% as $\lambda$ is changed from 0.001 to 0.02 jobs/s. As $\lambda$ increases, both $T$ and $P$ increase as well, which can be attributed to the higher contention for resources, and thus not all jobs are able to start executing at their earliest start times.

![Fig. 7. Effect of deadline of jobs on system performance.](image)

**Fig. 7. Effect of deadline of jobs on system performance.**

6) **Effect of the Number of Resources**

Fig. 9 shows that $T$ decreases as $m$, the number of resources for executing the jobs increases. The figure also shows that $O$ increases as $m$ decreases. When there are fewer resources, the CP Optimizer can find an initial feasible schedule quickly but it requires more time to refine the initial schedule such that $N$ is minimized. This is because the CP Optimizer has to explore whether matchmaking and scheduling different combinations of tasks on the limited number of resources can further reduce $N$. In addition, when $m$ is small, there are more tasks that cannot be scheduled to start executing at their earliest start times. This contributes to the increasing of $O$ for reasons similar to what was discussed in Section VI.B.3). There is no significant change to $O$ when $m$ is increased from 50 to 100 because in both cases a majority of tasks are able to execute at their earliest start times and do not have to wait. Further support for this is provided by the small increase in $T$ when $m$ is changed from 100 to 50. As expected, both $P$ and $T$ increase when $m$ is reduced from 50 to 25 because of the additional resource contentions occurring at smaller values of $m$.

![Fig. 8. Effect of job arrival rate on system performance.](image)

**Fig. 8. Effect of job arrival rate on system performance.**

![Fig. 9. Effect of the number of resources on system performance.](image)

**Fig. 9. Effect of the number of resources on system performance.**

VII. CONCLUSIONS AND FUTURE WORK

This paper describes an effective technique for resource management on clouds for jobs characterized by an end-to-end SLA comprising an earliest start time, execution time, and deadline, as well as multiple stages of execution. More specifically, we present a new constraint programming (CP) based technique and a resource manager, MRCP-RM, that can efficiently perform matchmaking and scheduling of MapReduce jobs with SLAs on a system that is subject to an open stream of job arrivals. MapReduce is used by many companies and institutions for a variety of applications including: large scale data processing and analysis (i.e. Big Data analytics), data mining, and scientific research (e.g. bioinformatics). More recently, MapReduce jobs requiring deadlines have become important for latency-sensitive applications such as live business intelligence [8]. Thus, it is important to have a resource management middleware that can efficiently perform matchmaking and scheduling of MapReduce jobs with deadlines, that is the focus of attention for this paper.

MRCP-RM is implemented in Java and uses IBM CPLEX to solve a CP formulation, which was devised to model the matchmaking and scheduling problem. One of the advantages of using CP is that for a given batch of MapReduce jobs, it is able to generate a job-to-resource mapping and schedule that leads to the minimum number of late jobs on the system. For the open system considered in this paper, the CP-based resource management algorithm is observed to achieve low values for the proportion of jobs missing their deadlines ($P$) when using a wide range of system and workload parameters. A comparison with the approach described in [8] called MinEDF-WC is provided. In addition, simulation-based factor-at-a-time experiments were performed to evaluate the effectiveness of MRCP-RM using a variety of system and workload parameters. A number of insights into system...
behaviour were gained by analyzing the experimental results, and are summarized next:

- For the system and workload parameters experimented with (see Table 3), the matching and scheduling overhead ($O$) was less than 0.05s in all cases except when: (1) $\lambda$ and $e_{\text{max}}$ are large, and (2) $d_M$ and $m$ are small. For systems with large $\lambda$, $e_{\text{max}}$, and small $d_M$, $O$ was less than 0.2s. When $m$ is small, $O$ was observed to be 0.57s.
- For all the experiments, $O$ was observed to be small in comparison to the average job turnaround time ($T$), which shows that MRCP-RM has a relatively low matchmaking and scheduling overhead: $O/T$ was less than 0.09% for the system and workload parameters presented in Table 3.
- The main factor that causes an increase in $O$ is the time it takes for the CP Optimizer to generate and solve the OPL model. In general, an OPL model that contains more tasks takes longer to generate and solve due to the higher number of constraints and decision variables that need to be processed.
- When increasing the job arrival rate (see Fig. 8), the scalability of the MRCP-RM algorithm in terms of $O$ was observed to follow a linear trend until a knee is reached. Even at 0.02 job/s (the highest value of $\lambda$ experimented with), $O$ is observed to be only 0.04% of $T$.
- $T$ is observed to increase most significantly in the cases where there are: high values of $\lambda$, small $m$, and large $e_{\text{max}}$. In all the other experiments, it was observed that $T$ was within 4% of the minimum job execution time.
- For most of the experiments discussed in this paper, MRCP-RM consistently generated a mapping and schedule with a $P$ of less than 0.6%, which is close to the lower bound of zero. In the scenarios where jobs have the highest chance to miss their deadlines: for small $d_M$, small $m$, high $\lambda$, and high $e_{\text{max}}$, $P$ was still observed to be low: 3.46%, 3.89%, 1.7%, and 1.96%, respectively.
- Comparison with MinEDF-WC[8]: It was shown that in comparison to MinEDF-WC, MRCP-RM can generate a mapping and schedule with a lower $T$, and a smaller $P$. A reduction in $P$ as high as 93% was observed when $\lambda=1/10000$ jobs/s whereas a 5% reduction in $T$ was observed in most cases (see Fig. 2 and Fig. 3). The overhead for running the MRCP-RM algorithm was observed to be small.

Overall, the experimental results showed that MRCP-RM can efficiently perform matchmaking and scheduling of MapReduce jobs with SLAs, while leading to a small proportion of jobs missing their deadlines, as well as a low matchmaking and scheduling overhead.

Directions for future research include the investigation of systems with additional resources including storage devices and communication links, as well as the consideration of monetary costs for resource usage. In addition, we plan to explore mechanisms that can reduce matchmaking and scheduling times when $\lambda$ is high. Generalization of the resource manager by incorporating capabilities for handling more complex workflows with user-specified precedence relationships warrants further investigation.

ACKNOWLEDGMENTS

We are grateful to Huawei Technologies Canada and the Government of Ontario for supporting this research.

REFERENCES