Data Throttling for Data-Intensive Workflows

Sang-Min Park and Marty Humphrey

Department of Computer Science, University of Virginia, Charlottesville, VA   22904

Abstract— **Existing workflow systems attempt to achieve high performance by intelligently scheduling tasks on resources, sometimes even attempting to move the largest data files on the highest-capacity links. However, such approaches are inherently limited, in that there is only minimal control available regarding the arrival time and rate of data transfer between nodes, resulting in unbalanced workflows in which one task is idle while waiting for data to arrive. This paper describes a data throttling framework that can be exploited by workflow systems to uniquely regulate the rate of data transfers between the workflow tasks via a specially-created QoS-enabled GridFTP server. Our workflow planner constructs a schedule that both specifies when/where individual tasks are to be executed,** *as well as when and at what rate data is to be transferred*. **Simulation results involving a simple workflow indicate that our system can achieve a 30% speedup when nodes show a computation/communication ratio of approximately 0.5. We reinforce and confirm these results via the actual implementation of the Montage workflow in the wide area, obtaining a maximum speedup of 31% and an average speedup with 16%. Overall, we believe that our data throttling Grid workflow system both executes workflows more efficiently (by better establishing balanced workflow graphs) and operates more cooperatively with unrelated concurrent Grid activities by consuming less overall network bandwidth, allowing such unrelated activities to execute more efficiently as well.**

I. INTRODUCTION

IN today's e-science collaborations, workflows play an increasingly important role by allowing scientists to describe the logic of scientific computational experiments in terms of tasks and dependencies between the tasks, often in the form of an intuitive diagram. The overall workflow-based infrastructure consisting of workflow planners, enactment engines, data movement tools, etc., hides the complex nature of underlying distributed resources and distributed system technologies from the domain scientists such that the scientists can concentrate on solving their problems without being distracted by the scale and complexity of infrastructure running the application. Many applications from such diverse domains as astronomy [1], physics [2], biology [3], and earthquake science [4] have recently used workflow languages and engines to successfully execute large-scale experiments on local resources and the Grid.

However, there are several obstacles that limit the workflow system's efficiency in the wide-area and diverse environment of the Grid. While sometimes the obstacles are found within the application – such as inherently limited parallelism because of the application structure, often times the reduced performance is due to the workflow engine that maps the abstract workflow to underlying resources in an inefficient manner. The first generation of Grid workflow engines were largely considered a success if they were consistently able to successfully execute *any* large workflow (not just the most efficient) across the uncertain Grid infrastructure, and recent efforts have been focused on generally scheduling component tasks more intelligently (e.g., [5]).

Perhaps the most significant open challenge of Grid workflows arises from the data-intensive nature of e-science workflows: individual tasks in the workflow can become bottlenecks as they

sit idle waiting for large amounts of data being produced by and/or delivered from other tasks. Some existing approaches attempt to take into account the data location and the link bandwidth information to attempt to avoid the data movement completely or move data via higher-capacity links whenever available [6][7]. However, the simplistic approach of moving the largest files (or the highest required data streaming rate) via the highest-capacity links can result in sub-optimal workflow execution:

- [INBOUND DATA] Even if the largest files are moved across the highest-capacity links, nodes can often wait for the *last* predecessor node to deliver its data before the node (by definition) can perform its intended functionality.

- [OUTBOUND DATA] Upon completing its execution, a task will often immediately move its output data to a successor node, with such data often arriving significantly before the receiving task can actually use it.

Intuitively, we desire that the *last* file arrive earlier, and the earliest-arriving files can actually arrive *later* if it were to some advantage to do so. If we were to achieve this, we could reduce or ideally eliminate such *unbalanced execution*, where some activity in the workflow is waiting for another activity. In fact, in general, the current practice of moving data from one location to another as early as possible is often either:

- *unnecessary* when viewed in isolation: any data that arrives before the last data arrives can be delayed, or
- *harmful* when viewed in-the-large: there is only finite capacity on each link, so multiple concurrent data movement operations can slow each other down significantly. Intuitively, more of the bandwidth should be used by the data movement for which a task waits.

This paper presents an end-to-end workflow system to minimize unbalanced computation in workflows for data-intensive e-science. The core of our approach is a unique data throttling framework that regulates the rate of data transfers between the workflow tasks via a specially-created QoS-enabled GridFTP server. Our workflow planner constructs a schedule that both specifies when/where individual tasks are to be executed, *as well as when and at what rate data is to be transferred*. Simulation results involving a simple workflow indicate that our system can achieve a 30% speedup when nodes show a computation/communication ratio of 0.5. To further evaluate our approach, we applied our workflow system to a well-known real Grid workflow (Montage). Across all runs, the least speedup was 11% and the most speedup with 31%, with an average speedup of 16%. Overall, we believe that our data throttling Grid workflow system both executes workflows more efficiently (by reducing unbalanced workflow graphs) and operates more cooperatively with unrelated concurrent Grid activities by consuming less overall network bandwidth, allowing such unrelated activities to execute more efficiently as well.

The rest of this paper is organized as follows. In Section 2 we present related work. Section 3 elaborates the concept and motivation of data throttling for workflow. The design of the throttling framework with the discussions on design decisions is presented in Section 4, followed by the presentation on the implementation in Section 5. Section 6 evaluates the framework and Section 7 concludes the paper with a discussion on future research.

II. RELATED WORK

Yu at al. provides a comprehensive survey of the existing workflow systems for e-science applications [8]. Pegasus [6] is a representative workflow system for e-science. It has been developed as a unified platform that allows scientists from diverse domains to describe the application logic in abstract way using a direct acyclic graph (DAG). The planner component of

Pegasus transforms the abstract DAG into concrete DAG that is executable on the distributed resources. It utilizes DAGMan and Condor-G [9] as an underlying enactment engine that coordinates the tasks and the data movement between the tasks. They run on top of GSI, GRAM, MDS, and GridFTP of Globus Toolkit [10] which provides the Grid-enabling functionality including authentication/authorization, remote execution, information monitoring, and data movement.

There have been efforts to enhance the performance of workflow execution. In Pegasus, the abstract DAG written by scientists goes through the process of clustering and partitioning by which the transformed DAG is made more efficient while the semantics remain the same. With clustering, the small tasks are grouped together as one executable unit such that the overhead of data movement and running the small tasks can be eliminated. Partitioning sets the scheduling *horizon* which determines which part of workflow will be scheduled at any given time. Similar approaches have been taken in [11] with further optimization at runtime by adapting to the dynamically changing state of underlying resources. Scheduling is a foundation of workflow optimization and there are works on the efficient scheduling of workflow applications on Grid [12][13]. Many of the scheduling algorithms for workflow use the heuristics for parameter-sweep applications [13] and enhance the heuristics with additional optimizing steps which consider the global structure of the workflow.

Nerieri et al. [14] presents the study of overhead analysis for Grid workflow applications. They found via a case study on two applications that load imbalance and data movement are among the very significant sources of overhead. Load imbalance occurs when some of parallel sections of workflow take longer to finish and make the workflow stall until the completion of the sections. This is a result of poor scheduling decision that could not make a balanced resource allocation. The data throttling plays an important role in coping with the load imbalance problem because the throttling can *hide the imbalance* by throttling up to the sections of parallel tasks that take longer to execute. The throttling can fix or complement the poor schedules at run time. We present this in more detail in the next section.

We have implemented an application level rate limiter as part of data throttling framework. The rate limiter implements the token bucket algorithm that is widely implemented in network routers [15]. Zhang et al. [16] uses the similar technique to implement an experimental QoS-aware GridFTP server in an application level. Globus Toolkit recently released the rate limiting XIO driver that uses the token bucket algorithm [17]. Though we use a similar overall approach to implement the rate limiting functionality, the architecture is different in many respects: our design is more centralized one in which there is a coordinator that manages multiple transfers and enforces the application's requirements on the transfer rate. The centralized management allows the coordinator to rapidly adapt to changes in the observed transfer rate in order to keep the application's requirements satisfied at all times. Furthermore, we report the novel use of application level rate limiting for workflow applications.

III. DATA THROTTLING FOR WORKFLOW

We define the general models of workflow in the form of a DAG as depicted in Figure 2. The workflow is expressed as a set of tasks with data dependencies between them. The node and edge in a DAG represents the task and the data dependencies respectively. A task becomes executable when all of its inputs are ready. There are two types of workflow -- abstract and concrete workflow. An abstract workflow is written by the workflow programmer (e.g., scientists) and describes the application's logic at a high level of abstraction. It becomes a concrete workflow

after a scheduler maps the tasks to resources. *Makespan* is a term referring to the performance metric of workflow and defined as the interval between the start time of first task and the end time of final task. We use the example workflow in Figure 1 throughout this Section. Figure 1 is an abstract workflow. We will identify the resource binding explicitly if it is to be interpreted as a concrete workflow.
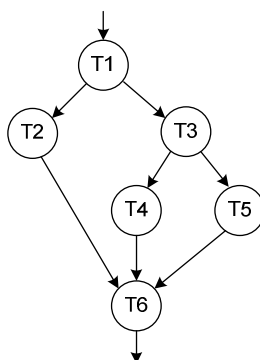

Figure 1. Representative Workflow

Two of the characteristics of workflow that motivate the data throttling are **load imbalance** and **synchronization**. (The third motivation – generally to consume as little bandwidth as possible without compromising performance, so that more bandwidth is available to unrelated Grid activities – is not discussed in this section). Load imbalance occurs when some of workflow's branches take longer to finish than the others. For example, load imbalance in Figure 1 can cause the synchronization point (T6) to stall until all of the branches have completed the tasks (T2, T4, T5). There are the two main sources of load imbalance. One is a structural imbalance that can exist due to the unbalanced structure of *abstract* workflow. Figure 1 has a structural imbalance as paths from T1 to T6 have different depths (assuming all tasks require the same amount of time). Some applications inherently reveal the unbalanced structure in its flow of logic. For instance, the physics application LIGO [2] has a highly unbalanced workflow structure. The other type of imbalance, called schedule imbalance, is due to the unbalanced resource allocation made by the scheduler. The scheduling heuristics in general attempt to allocate resources to a set of tasks in a way that minimizes the imbalance among the branches. However in many cases, the scheduler fails to produce the balanced schedule due to limited resource availability and /or coarse granularity of individual tasks. For example, if T4 takes twice as long as T5, and the available resources are all homogeneous, the scheduler has no choice but to let the imbalance occur. Our data throttling capability plays an important role in helping to reduce load imbalance. For example, assume all tasks in Figure 1 are scheduled on different resources and take the same execution time on the scheduled resources. As soon as T1 produces the output, T2 and T3 fetch the output from T1 and the two simultaneous transfers occur on the shared link at the resource T1. We further assume that all the resources have the same physical bandwidth without congestion. Without our data throttling mechanism, which task receives T1's output first will depends on the TCP congestion control algorithm at T1 – it is likely that T2 and T3 will receive the data at roughly the same time because many TCP congestion control algorithms utilize a fairness metric [17]. By using our data throttling mechanism in our modified GridFTP, we are able to allocate a higher bandwidth for the transfer to T3, and the makespan can be reduced by as much time that is saved by transferring data to T3.

Data throttling is also productive when multiple workflows are run on a shared resource. For example, many scientific collaborations have a relatively small number of data servers for storing the raw and output data, and many instances of workflow are simultaneously executed, continuously fetching raw data from the server and storing the output data to the server. For instance, in Figure 1, T6 if often a task that synthesizes data from many workflow branches into a single file and stores it into a data server. Assume that the two instances of T6, called $T6_1$ and $T6_2$, run on the data server for the different instances of the workflow. Furthermore assume that data between $T2_1$ and $T6_1$ is transferred at a low bandwidth (e.g., due to congestion on the resource running $T2_1$) compared to $T4_1$- $T6_1$ and $T5_1$- $T6_1$. Even if the high bandwidth between the two latter paths allows faster data movement, there is no advantage of moving the data quickly because $T6_1$ can start only after data from $T2_1$ is received. This workflow will be delayed anyway – from the point of global optimization, it is better to concede the unnecessarily high bandwidth at $T4_1$- $T6_1$ and $T5_1$- $T6_1$ to another workflow instance. In other words, we need to **throttle down** the unnecessarily high bandwidth to benefit the concurrently executing workflow instance.

One may notice that the explanations above are based on the contradictory assumption about the property of underlying network (no congestion vs. congestion). However, the assumptions about network are only for the easier explanation and do not contradict the argument. In fact the motivation for *throttling down* can be found in the case of load imbalance. As an illustration, we begin by throttling up T1-T3 (and throttling down T1-T2 to give a room to throttle up) for hiding the load imbalance. If we later find congestion on T1-T3 causes the low bandwidth than the point we set to throttle up, we throttle down the T1-T2 further in order not to waste the bandwidth. In summary, if the network link has unlimited bandwidth (theoretically), data throttling moves data to the tasks that needs the data more urgently than the other. On the contrary if the congestion limit the bandwidth for some transfers, data throttling reduces the unnecessarily wasted bandwidth that may otherwise be used by the other applications.

IV. DATA THROTTLING FRAMEWORK

The Data Throttling Framework consists of two components: the **Workflow Engine** and the **Rate Throttler**. Our workflow engine is unique in that it considers a new attribute that we refer to as the **Relative Transfer Time Finished (RTTF)**. *RTTF* is a tuple in which the values correspond to edges and represent the desirable time at which data transfer is finished. There are two types of *RTTF* per task, one for the group of incoming edges and the other for the outgoing edges. More formally *RTTF* is defined as follows.

For a given Directly Acyclic Graph G = {V, E} where V is the set of nodes (tasks) and E is the set of edges (data dependencies),

$$RTTF_I(v) = \{f(e_1), f(e_2), \ldots, f(e_n) \mid e_i = (v_i, v) \in E \text{ for all } v_i \in V\}$$

$$RTTF_O(v) = \{f(e_1), f(e_2), \ldots, f(e_n) \mid e_i = (v, v_i) \in E \text{ for all } v_i \in V\}$$

We define $T(e_i)$ as a time that the file transfer is finished from v_i to v_j for $e_i = (v_i, v_j)$.

The function f is defined such that $f(e_x)/f(e_y) = T(e_x)/T(e_y)$. *RTTF* value for an edge is relative to the other edges only within the same group of incoming and outgoing edges.

As the above definition implies, the *RTTF* captures the relative order of time by which the transfers are finished. Using the *RTTF*, the workflow programmer or workflow engine can set a

requirement about the data transfer delay. If the task does not have *RTTF* annotated to it, there is no such requirement for the flow from/to that task. The *RTTF* is set in order to prevent a load imbalance or an otherwise overly-aggressive consumption of bandwidth. For example, in Figure 1, the *RTTF* of T1 and T6 can be set to *RTTF$_O$(T1)={2, 1}* and *RTTF$_I$(T6)={1, 1, 1}*. T1's *RTTF* for outgoing edges tries to compensate the structural load imbalance by describing that T3 should receive its output in half the time as compared to T2. T6's RTTF specifies that all of the incoming transfers should arrive at the same time (in order to keep from unnecessary surge of bandwidth use).

It is possible that a workflow programmer annotates the DAG with *RTTF* values by himself/herself, but it is more likely that the workflow engine performs that task for the following reasons:

- The abstract workflow written by a programmer can be restructured by the workflow planner, and thus the RTTF may not be correct when the restructuring is done.
- Data throttling is used for hiding the load imbalance and the expected amount of load imbalance can be known after the scheduler allocates tasks to resources.

Note that *RTTF* does not specify the bandwidth requirements for transfers, but rather specifies the requirements in terms of time. Specifying bandwidth requirements is an alternate approach, which was not chosen due to implementation complexity and ease of use. The file size must be known before the bandwidth can be derived from time requirement (i.e., Bandwidth=DataSize / Time). From RTTF, the workflow engine generates the *Relative Transfer Bandwidth (RTB)* as follows:

$$RTTF(v) = \{r_1, r_2, \ldots, r_n\}$$

$$Inversed\ RTTF(v) = \left\{\frac{1}{r_1}, \frac{1}{r_2}, \ldots, \frac{1}{r_n}\right\}$$

$$Relative\ Transfer\ Bandwidth(v) = \left\{\frac{fs(e_1)}{r_1}, \frac{fs(e_2)}{r_1}, \ldots, \frac{fs(e_n)}{r_n}\right\}\ where\ fs(e_i)\ refers\ to\ the\ size\ of$$

file transferred from v_i to v_j and $e_i = (v_i, v_j)$

When the workflow engine (enactor) is ready to submit a task to a scheduled resource, it presents the *RTB* of the task to the Rate Throttler that runs on the scheduled resource. Upon receiving the RTB, Rate Throttler translates it to the concrete bandwidth allocation for each transfer and enforces the allocated bandwidth. The concrete bandwidth allocation from RTB depends on the particular technology implementing the Rate Throttler functionality. Although we implement the Rate Throttler using a token bucket rate limiting in an application level, RTB is appropriate for a range of implementation approaches – for example by setting up the connection-oriented virtual circuits between the resources [19]. The functionalities the Rate Throttler should be capable of are:

- The Rate Throttler must implement the interface that accepts *RTB,* and a translation routine must exist to converts the *RTB* to a concrete bandwidth allocation
- It must enforce the bandwidth while keeping the meaning of values in RTB satisfied
- It should adapt to the changes of the network state (e.g., over-congestion) such that the dynamic state change does not result in violation of relative bandwidth

We have found that this design decision – separating the roles of Workflow Engine and Rate Throttler and defining a clean interface between them using *RTB* – has many benefits. The engine does not need to care about the working mechanisms of Rate Throttler, but it only describes the requirements in terms of relative time which is of relevant to it. Since the bandwidth requirements delivered to Rate Throttler is in terms of relative bandwidth, Rate Throttler can determine how

much share of its capacity to allocate to the group of edges and assign bandwidths within that share. It is robust since workflow engine can continue to use resources where there is no Rate Throttling support or Rate Throttler is broken. The engine can ignore the *RTTF* of a task and continue to run the workflow. It may not benefit from the service provided by the data throttling framework, but still the workflow can be executed on resources with perhaps more overhead. Allocating concrete bandwidth from the relative bandwidth requirements is easy to implement if the underlying network QoS service supports bandwidth provisioning. The framework does not require significant changes to the existing workflow engine; only few additional annotations on DAG grammar and its interpretations are sufficient.

V. DATA THROTTLING FRAMEWORK – IMPLEMENTATION

In this section, we describe the implementation of the Data Throttling Framework.   The Workflow Engine is written in Java and accepts the workflow (DAG) in XML. Our implementation allows the programmer or engine to add *RTTF* annotations to the nodes of DAG. The *RTTF* is translated to RTB and delivered to the resources at run-time (note that not all nodes have an *RTTF* annotation). In this section, we give an overview of the entire system with an emphasis on how the annotations are created and used.

We implemented the Rate Throttler component as an application-level rate limiter. Rate Throttling requires that one can dynamically determine and allocate the available bandwidth on a per-file-transfer basis. Rate limiting is a foundation for providing rate-based QoS because we need to limit the rate of non-QoS client's transfers in order to provide the constant bandwidth to a QoS client. In order to "throttle up" the bandwidth of some transfers, we need a mechanism that "throttles down" the other transfers sharing the link. Traditionally, network QoS has been provided as a link- and network-layer protocol, and the bandwidth provisioning is implemented in routers. However, while in-network QoS provisioning such as DiffServ [19] is widely implemented, the end users still have no control over the bandwidth allocation. Recently connection-oriented circuit switching has emerged as a novel way of constructing QoS network that allows end-users to control the bandwidth allocation [18]. While circuit-switched networking and/or optical networks have the potential to provide a solution to this problem in future networks, this doesn't solve the problem for today's Internet.

Building QoS at the application level generally assumes that the backbone and intra-network is free of congestions and the only point that congestion occurs is on the end-host. Although this claim cannot hold true always, it is surprisingly true in many circumstances. Especially the e-science community has relied on the academic/research backbone which has tens of gigabit/s capacity (e.g., NSF TeraGrid network link at 40Gb/s and Internet 2 at 10 Gb/s). The rapid increase in the backbone and slow upgrade of local network has increasingly pushed up the bandwidth pressure on end-hosts especially on the hosts running popular service (e.g., Web and GridFTP). We repeatedly confirmed this phenomenon (no congestion on backbone and congestions at end host) while we tested the rate limiter for the transfers over Internet 2 backbone.

We implemented the application-level rate limiter using the well-known token-bucket rate limiting. The token-bucket or similar algorithms such as leaky bucket is implemented in routers for rate-limiting purposes. In the token bucket model, when packets of n bytes arrive to the router, they are forwarded to the next hop only if there are n or more number of tokens in the bucket. The n tokens are removed while the packet is forwarded. If there are less than n tokens, the router waits for the tokens to be filled while queuing or discarding the packets. The bucket has height H and the token is added to the bucket at rate R, only if the number of tokens in the bucket does not

exceed the bucket height. Thus, *H* is the parameter that determines peak burst rate, and *R* determines the average rate. The target transport service that we apply the token bucket rate limiting is GridFTP, one of the most widely used transport protocol and software for high performance data transfer. Many existing workflow engines use the GridFTP as a data movement tool. We implemented an XIO driver that replaces the default TCP XIO driver of GridFTP server [21]. The XIO driver receives the token from the rate limiter and transfers a block of data as long as the token remains in the bucket. If no token is available, the XIO driver stalls until enough tokens are received from the rate limiter. Note that we only replace the XIO driver on server side, and the GridFTP client needs no modification. Figure 2 illustrates the overview of the Rate Throttler.
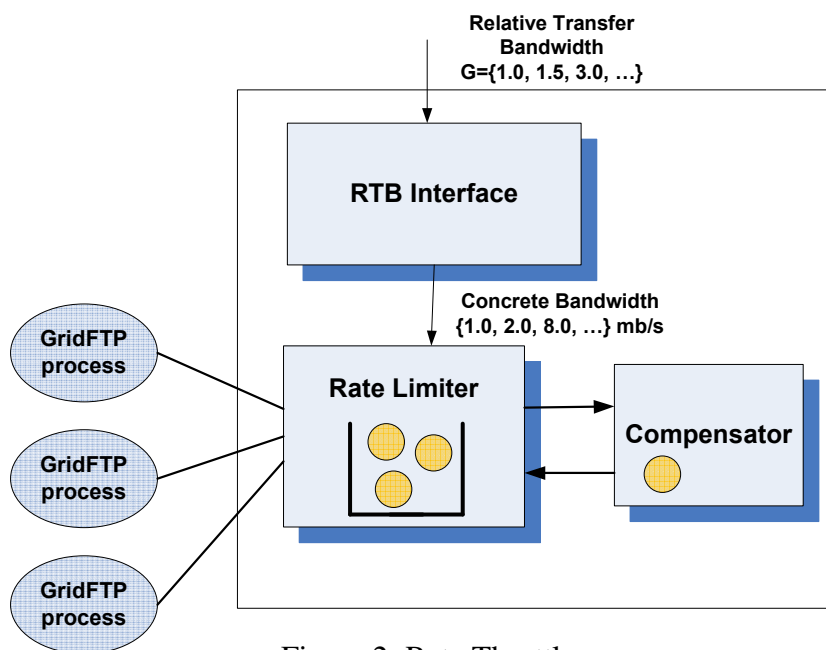
Figure 2: Rate Throttler

In Figure 2, the compensator component complements the rate limiter in meeting the bandwidth target. With only the rate limiter, the flow's bandwidth become less than the rate limit target since TCP exhibits a frequent bandwidth drop on a congested path (as a result of TCP's reduced window size for a packet drop event). Even though the long term average bandwidth along the paths (without rate limit) is above the rate limit target, setting the limit on target *x* may result in average bandwidth less than *x* because the flow's rate drops at some instants. Setting the rate limit target higher than the allocated bandwidth may overcome this problem, but can result in higher average bandwidth than necessary. The compensator copes with this problem by monitoring the rate of existing transfers and issuing extra tokens to the transfer that suffers from sudden bandwidth drops. The compensator does this until the average bandwidth of the transfer is recovered to a rate limit target. The Rate Throttler assigns portions of capacity (e.g., 10%) to the compensator for allowing an extra space to overcome bandwidth drop.

The RTB received from the workflow engine is translated to a concrete bandwidth allocation for each transfer. An administrator specifies the capacity of the link on which the Throttler runs (e.g., 100 mb/s), and the RTB Interface monitors and maintains the RTB requests it has received from the external workflow engines. The concrete bandwidth allocated to a RTB is the link

capacity divided by the number of RTBs. The simple formula to derive the concrete bandwidth from RTB is as follows:

{c.b$_1$, c.b$_2$, …, c.b$_n$} = BW$_{RTBi}$*{r.b$_1$, r.b$_2$, …,r.b$_n$} where r.b$_i$ and c.b.$_i$ denotes the relative and concrete bandwidth of edge i, and BW$_{RTBi}$ refers to the bandwidth allocated for RTB $_i$.

VI. EVALUATION AND DISCUSSION

In this section, we evaluate the developed Data Throttling Framework in two ways. First, we assess the range of applications that will benefit from the framework by performing simulation studies involving workflows with varying ratios of computation to communication. Second, in part to validate the simulation results, we apply the Data Throttling Framework to a real astronomic workflow application, Montage [1].

A. *Effective Computation-Communication Ratio*

Not every workflow application will benefit from the Data Throttling Framework. If the amount of data flowing between tasks is too small relative to the duration of a computation, adjusting the data-transfer rate will have no effect on the makespan. Conversely, if the execution time of tasks is too short compared to a large communication delay, it will not be beneficial to throttle the data rate since the task imbalance is negligible. Thus, there is a range of application's Computation-to-Communication Ratio (CCR) that will benefit from data throttling. We identify the range via simulation. We implemented the functions of workflow engine and Rate Throttler on top of GridSim [22], a Java-based, discrete event simulation package for Grid applications. The network package in GridSim provides a packet-level flow modeling and modeling of DiffServ [23] routers in which routers enforce stream's flow rate with packet's type of service field. We utilize this capability to implement the Rate Throttler. The Rate-QoS is able to be set in the routers, and the Rate Throttler configures them at runtime. The simulated workflow engine and rate throttler works in the way we presented previously. In the simulated engine, *RTTF* is determined using simple heuristics; the workflow engine estimates the task's expected execution time on the resources and derives a relative time that is the inverse of the estimated execution time.

We run the simple workflow depicted in Figure 3 to gain insight on the effect of CCR to data throttling. We chose to study the relatively simple workflow of Figure 3 to better understand and quantify our system's behavior, noting that such a pattern is likely to occur in larger, more complex workflows. We do not claim the data throttling would work for every parts of large workflow; Simulation results in this subsection will help identifying the parts.

In our simulation studies, the head node (H) distributes one output file to *n* parallel nodes (P), and the output files of parallel nodes are merged at the tail node (T). We define the CCR of this workflow as $\frac{\text{execution time of p}_i}{\text{communication time of p}_i}$ and the communication time of P$_i$ is determined by the factors including H's output file, the bandwidth between H-P and P-T, the number of P nodes, and the output of P$_i$. A CCR of P$_i$ equaling one indicates that the time to move data through H – P$_i$ – T is the same as P$_i$'s execution time. In order to study and quantify a load imbalance between parallel nodes, we vary the execution time of P$_i$ and assume all P$_i$ run on homogeneous resources. The execution time of P$_i$ is randomly generated within the range of {base, base$\pm$(variance)*base} where *base* is equivalent to the given CCR. As the variance becomes large, the load imbalance grows. We run simulations with varying degree of CCR and for different variances on load imbalance. We arbitrarily fixed the execution time of H and T at 1/10

of P_i's average execution time. The output size of P_i is assumed to be very small (compared to input to P_i). We run the simulation with baseline where there's no Rate Throttler running on resources, and with Rate Throttler. Figure 4 presents simulation result, measuring the Speedup, defined as $\frac{\text{makespan of baseline}}{\text{makespan with Data Throttler}}$. Each number in the result is average of 5 runs. The simulation result shown in Figure 5 is bell-shaped, indicating that as the computation and communication ratio is between 0.5 and 1, the data throttling is most effective. The variance of load imbalance makes a large impact on speedup in the range {CCR < 1}, but does a little impact on the range {CCR>=1}
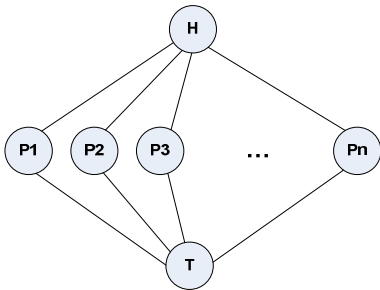
.

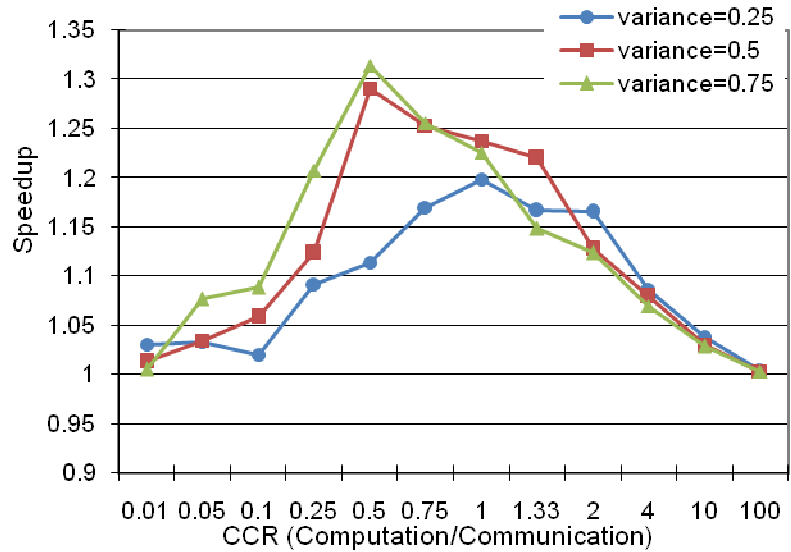

Figure 3: Simple Workflow   Figure 4: Speedup vs. CCR

B. Montage Workflow

Montage [1] is an astronomical application by which multiple image sets containing the photometric observation of celestial objects in various spectrum located at overlapping regions of sky (i.e., mosaics) are integrated into a single image file. Montage is both compute- and data-intensive and exhibits a CCR within the range that was shown to be effective in simulation studies. It executes in several steps, and the compute-intensive steps can be run in parallel without communication between independent tasks. For this reason Grid workflow systems such as Pegasus [6] have been used to run the parallelized version of Montage on large scale Grid resources. We program Montage as a workflow using our DAG language introduced in Section 4. The workflow engine runs the tasks in the order programmed in Montage's DAG. Figure 5 illustrates the Montage workflow. The circle in the figure represents a parallel task that runs on distributed resources and the rectangle illustrates the synchronizing tasks that gathers output from previous level's tasks and prepares for the input for the next level parallel tasks. While the solid line represents the movement of large image files, the dotted line represents the movement of metadata which are typically in few kilobytes. In Table 1 we present the more detailed characteristics of Montage by showing the size of example data movement and execution times

when it runs on single node or in parallel. A speedup of about 3.2 can be expected when it runs on four local nodes.
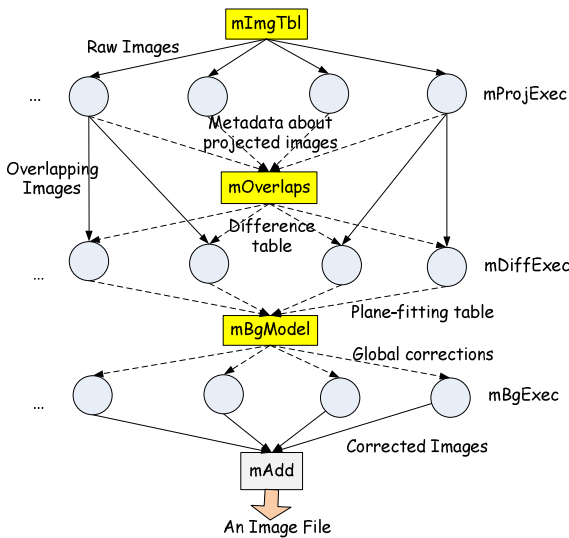

Figure 5: Montage Workflow

Table 1: Example Montage Execution

Component	Exec. Time or Data Size to Move	
	1 node	4 nodes
Raw Images	0	180 MB
Overlapping Images (total)	0	720 MB
Corrected Images (total)	0	720 MB
mImgTbl	5	5
mProjExec	280	70
mOverlaps	1	1
mDiffExec	80	20
mBgModel	5	5
mBgExec	32	8
mAdd	20	20
Total task execution time	423 sec	129 sec

In our experiment, the resources to run Montage are located at University of Virginia (UVa) and NCSA [24]. The data server running at UVa stores the *"Two Micron All Sky Survey (2MASS)"* images acquired from NASA/IPAC Infrared Science Archive [25]. Tasks are statically scheduled onto resources since resources are predetermined and available for all times in this experimental setup. The synchronizing tasks (rectangle) except for *"mADD"* are executed on UVa data server and the parallel tasks (circle) are executed on four nodes at NCSA. We run the synchronizing tasks at the data server since the data server's CPU is idle and the input to the synchronizing tasks is very small. However, the large input (i.e., corrected images) of *"mADD"* prevents us from running it on data server. The movement of two sets of large data, overlapping images and corrected images, can be done relatively quickly since data is exchanged through gigabit LAN in NCSA. The only data that takes substantial amount of time to deliver is raw images from UVa to NCSA, and this is the part data throttling engages in. The two sites UVa and NCSA are connected via Internet 2 backbone with a 10 Gb/s capacity. More detailed specifications of resources are summarized in Table 2.

Table 2: Resource Specification

	UVa	NCSA
Number of Node	1	4
CPU/RAM/IO	Intel Core2 6400 Dual (2.13 GHz) 2 GB RAM PCI-Express	2 Intel Xeon (3.0 GHz) 2 GB RAM
Disk	4-way SATA RAID 0 (Striping)	Single SATA
Local Network	Fast Ethernet (100 mb/s)	Gigabit Ethernet
TCP Setting	BIC congestion control TCP autotuning (16 MB buffer)	BIC congestion control TCP autotuning (16 MB buffer)
O/S	Linux 2.6 kernel	Linux 2.6 kernel

The Rate Throttler runs on the data server at UVa, adjusting the rate of raw image transfer to the nodes at NCSA according to the RTB (Relative Time Bandwidth) values dynamically received from the workflow engine. In this experiment workflow engine runs on a separate machine at UVa.

Note that the four nodes at NCSA are homogeneous. In order to evaluate the framework under the presence of a task imbalance situation, we made one of node execute *'mProjExec'* twice. Under this assumption, the workflow programmer requests rate requirements with RTTF values {1.0, 2.0, 2.0, 2.0} specifying that the delivery of raw images to the slow node (i.e., node running *'mProjExec'* twice) should be done in half the time of the other nodes. This is based on a simple assumption that the CCR is one. We run Montage with 10 different sets of mosaic images under the baseline and with data throttling. In the baseline, the unmodified GridFTP runs at UVa while with data throttling the GridFTP's data transfer rate is controlled by Rate Throttler. The result of this experiment is presented in Table 3. From the table, we can see there are 30-80 seconds of difference of *makespan* between the baseline and the data throttling. All of the 10 runs have achieved speedup ranging from 1.11 to 1.31 and average speedup is 1.16.

Table 3: Results from Running Montage with 10 Different Images

2Mass object	Baseline (sec.)	Rate Throttled (sec.)	Speedup (Baseline/R.Th.)
M70	253.3	219.5	1.15
M71	274.3	226.5	1.21
M72	349.5	266.8	1.31
M73	307.5	277.5	1.11
M74	267.6	234.3	1.14
M75	294.0	256.3	1.15
M76	299.5	268.6	1.12
M77	319.7	282.8	1.13
M78	276.4	242.5	1.14
M79	296.1	263.6	1.12
Average	293.8	253.8	1.16

Under the baseline case, the transfer rate of raw images is governed by uncontrolled competition among the multiple TCP streams to NCSA nodes. This results in unpredictable time at which raw images are ready to be consumed by parallel tasks. However with data throttling, the Rate Throttler adjusts the transfer rate as dictated by RTTF values, providing more bandwidth to the tasks with smaller RTTF value (1.0) and suppressing the rate of other transfers with the higher RTTF value (2.0). With rate throttling, the raw images are delivered earlier to the slow node. The variations of speedup among the 10 runs are due to variations of transfer time in the baseline case. In the baseline case, the raw image's delivery time is not predictable. In some executions, the raw images happens to be delivered faster to the slow execution node resulting in small speedup, while in other cases it can be delivered later to the slow execution node making the speedup bigger. Table 4 presents a more detailed comparison of the timeline of the executions' steps. The table contains the average execution time of 10 image sets at various stages during the workflow's progress. With data throttling, we can see that the raw images are delivered to slow execution node 2.4 times faster than the other nodes, and this compensated for the slow execution of '*mProjExec*' resulting in smaller difference of *mProjExec*'s finish time between nodes (67 sec vs. 31 sec). While the faster delivery of raw images to the slow node slightly increased the delivery time of raw images to other nodes (5.5 seconds) it does not impact to the workflow's makespan since the makespan is limited by the slow node.

Table 4: Comparison of Execution Time at Stages of Montage

Steps	Baseline (sec.)	Rate Throttled (sec.)	Difference (sec.)
mImgTbl finished	6.31	5.27	1.04
Raw img. delivered to slow node	56.08	24.77	31.31
Raw img. delivered to other nodes	54.67	60.17	-5.5
mProjExec finished at slow node	191.82	161.65	30.17
mProjExec finished at other nodes	124.27	130.20	-5.93
mOverlaps finished	203.75	169.10	34.65
mBgModel finished	265.86	226.54	39.32
mAdd finished	293.8	253.8	40.0

VII. CONCLUSION

Existing workflow systems attempt to achieve higher performance by intelligently scheduling tasks on resources, taking into account the bandwidth into and out of individual compute nodes. However, such approaches are limited, in that there is still only limited control available regarding the arrival time and rate of data transfer between nodes.

In this paper, we designed and implemented new capabilities for higher efficiency and balance in Grid workflows by creating a data throttling framework that allows a workflow programmer/engine to describe the requirements on the data movement delay. They can utilize this tool to balance the execution time of workflow branches and eliminate unnecessary bandwidth usage, resulting in more efficient execution. We presented the framework design that separates the concerns of the workflow system from the underlying network QoS providers. We evaluated the applicability of the data throttling framework in simulation, and reinforced and confirmed these results via the actual implementation of the Montage workflow in the wide area, obtaining a maximum speedup of 31 % and an average speedup with 16%.

In future work, we plan to further explore the tradeoffs involved in choosing the parameters of the data throttling utilized in our system. Specifically, we plan to automatically derive the precise data delay requirements from the workflow schedule and experiment with a wider range of actual Grid workflows.

REFERENCES

[1] G. B. Berriman, E. Deelman, J. Good, J. Jacob, D. S. Katz, C. Kesselman, A. Laity, T. A. Prince, G. Singh, and M. Su. Montage: a Grid Enabled Engine for Delivering Custom Science-Grade Image Mosaics on Demand. Proceedings of the SPIE Conference on Astronomical Telescopes and Instrumentation, June 2004.

[2] Duncan A. Brown, Patrick R. Brady, Alexander Dietz, Junwei Cao, Ben Johnson, and John Mc-Nabb. A Case Study on the Use of Workflow Technologies for Scientific Analysis: Gravitational Wave Data Analysis. Workflows for eScience, (Springer-Verlag, 2006).

[3] T. Oinn, M. Addis, J. Ferris, D. Marvin, M. Senger, M. Greenwood, T. Carver and K. Glover, M.R. Pocock, A. Wipat, and P. Li. Taverna: a tool for the composition and enactment of bioinformatics workflows. Bioinformatics, 20(17):3045-3054, Oxford University Press, London, UK, 2004.

[4] Deelman, E., Callaghan, S., Field, E., Francoeur, H., Graves, R., Gupta, N., Gupta, V., Jordan, T. H., Kesselman, C., Maechling, P., Mehringer, J., Mehta, G., Okaya, D., Vahi, K., and Zhao, L. 2006. Managing Large-Scale Workflow Execution from Resource Provisioning to Provenance Tracking: The CyberShake Example. In Proceedings of the Second IEEE international Conference on E-Science and Grid Computing, December 2006

[5] Blythe, J., Jain, S., Deelman, E., Gil, Y., Vahi, K., Mandal, A., Kennedy, K. Task scheduling strategies for workflow-based applications in grids. Cluster Computing and the Grid, 2005. CCGrid 2005. IEEE International Symposium on , vol.2, no., pp. 759-767 Vol. 2, 9-12 May 2005

[6] Ewa Deelman, Gurmeet Singh, Mei-Hui Su, James Blythe, Yolanda Gil, Carl Kesselman, Gaurang Mehta, Karan Vahi, G. Bruce Berriman, John Good, Anastasia Laity, Joseph C. Jacob, Daniel S. Katz. Pegasus: a Framework for Mapping Complex Scientific Workflows onto Distributed Systems. Scientific Programming Journal, Vol 13(3), 2005, Pages 219-237

[7] Rubing Duan, Radu Prodan and Thomas Fahringer. Run-time Optimization for Grid Workflow Applications. In International Conference on Grid Computing, IEEE Computer Society Press, Barcelona, Spain, September 2006.

[8] Jia Yu and Rajkumar Buyya. A Taxonomy of Workflow Management Systems for Grid Computing. Journal of Grid Computing, Volume 3, Numbers 3-4, Pages: 171-200, Springer Science Business Media B.V., New York, USA, Sept. 2005.

[9] Douglas Thain, Todd Tannenbaum, and Miron Livny. Distributed Computing in Practice: The Condor Experience. Concurrency and Computation: Practice and Experience, Vol. 17, No. 2-4, pages 323-356, February-April, 2005.

[10] I. Foster. Globus Toolkit Version 4: Software for Service-Oriented Systems. IFIP International Conference on Network and Parallel Computing, Springer-Verlag LNCS 3779, pp 2-13, 2006.

[11] Rubing Duan, Prodan, R., Fahringer, T. Run-time Optimisation of Grid Workflow Applications. 7th IEEE/ACM International Conference on Grid Computing, Sept. 2006

[12] Mandal. A, Kennedy. K, Koelbel. C, Marin. G, Mellor-Crummey. J, Liu. B and Johnsson. L Scheduling Strategies for Mapping Application Workflows onto the Grid. IEEE International Symposium on High Performance Distributed Computing (HPDC 2005).

[13] Tracy D. Braun et al. A comparision of eleven static heuristics for maping a class of independent tasks onto heterogeneous distributed computing systems. Journal of Parallel and Distributed Computing, 61:810–837, 2001.

[14] Nerieri, F., Prodan, R., Fahringer, T., Hong-Linh Truong. Overhead Analysis of Grid Workflow Applications. 7th IEEE/ACM International Conference on Grid Computing, Sept. 2006

[15] Bernet, Y., Blake, S., Grossman, D., Smith, A. An Informal Management Model for Diffserv Routers. RFC 3290.

[16] Zhang, H., Keahey, K., Allcock, W. Providing Data Transfer with QoS as Agreement-Based Service. pp. 344-353, 2004 IEEE International Conference on Services Computing (SCC'04), 2004.

[17] http://www.globus.org/toolkit/docs/development/4.1.0/common/xio/xio-release-notes.html

[18] Lisong Xu, Khaled Harfoush, and Injong Rhee. Binary Increase Congestion Control for Fast Long-Distance Networks. INFOCOM 2004.

[19] X. Zheng, M. Veeraraghavan, N. S. V. Rao, Q. Wu, and M. Zhu. CHEETAH: Circuit-switched High-speed End-to-End Transport ArcHitecture testbed. IEEE Communication Magazine, vol. 43, Issue 8, pp. s11-s17, Aug. 2005.

[20] Blake, S., Black, D., Carlson, M., Davies, M., Wang, Z., Weiss, W. An Architecture for Differentiated Services, RFC 2475 (1998).

[21] Allcock, W., Bresnahan, J., Kettimuthu, R., Link, J. The Globus eXtensible Input/Output System (XIO): A protocol independent IO system for the Grid. Proceedings of the Joint Workshop on High-Performance Grid Computing and High-Level Parallel Programming Models held in conjunction with International Parallel and Distributed Processing Symposium (IPDPS 2005), April 2005.

[22] Anthony Sulistio, Gokul Poduval, Rajkumar Buyya, and Chen-Khong Tham, On Incorporating Differentiated Levels of Network Service into GridSim, Future Generation Computer Systems (FGCS), ISSN: 0167-739X, Volume 23, Issue 4, May 2007, Pages: 606-615 Elsevier Science, Amsterdam, The Netherlands, May 2007.

[23] Blake, S., Black, D., Carlson, M., Davies, M., Wang, Z., Weiss, W. An Architecture for Differentiated Services, RFC 2475 (1998).

[24] National Center for Supercomputing Applications: http://www.ncsa.edu

[25] http://irsa.ipac.caltech.edu/applications/2MASS/IM/