Future Generation Computer Systems 25 (2009) 904–911

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Flow scheduling and endpoint rate control in GridNetworks

Sebastien Soudan [a,*], Bin Bin Chen [b], Pascale Vicat-Blanc Primet [c]

[a] Université de Lyon, LIP, École Normale Supérieure de Lyon, France
[b] Department of Computer Science, National University of Singapore, Singapore
[c] INRIA, Université de Lyon, LIP, École Normale Supérieure de Lyon, France

ARTICLE INFO

Article history:
Received 9 December 2007
Received in revised form
3 June 2008
Accepted 17 June 2008
Available online 3 July 2008

Keywords:
Grid networks
Flow scheduling
Bulk data transfers
Rate limitation
Pacing

ABSTRACT

In grid networks, distributed resources, computing or storage elements as well as scientific instruments are interconnected to support computing-intensive and data-intensive applications. To facilitate the efficient scheduling of these resources, we propose to manage the movements of massive data set between them. This paper formulates the bulk data transfer scheduling problem and presents an optimal solution to minimize the network congestion factor of a dedicated network or an isolated traffic class. The solution satisfying individual flows' time and volume constraints can be found in polynomial time and expressed as a set of multi-interval bandwidth allocation profiles. To ensure a large-scale deployment of this approach, we propose, for the data plane, a combination of a bandwidth profile enforcement mechanism with traditional transport protocols. The paper examines several solutions for implementing such a mechanism in a Linux kernel. The experimental evaluation shows that packet pacing performed at IP level offers a simple yet valuable and TCP-compatible solution for accurate bandwidth profile enforcement at very high speed.

1. Introduction

Grid computing is a promising technology that brings together geographically distributed resources to build very high performance computing environments for data-intensive or computing-intensive applications. In such grids, massive quantities of data (terabytes or even petabytes) are expected to be produced and distributed around the world for analysis and processing. Therefore, these infrastructures are often relying on dedicated high speed optical networks. These networks present theoretical aggregation factors and multiplexing degrees much lower than those observed in traditional Internet context. Indeed, the aggregation factor, which is the access link's capacity over the nominal capacity of end nodes links, is about 1 or 10 due to the high capacity of the network interface of the interconnected servers and the relatively modest capacity of the access links. In xDSL or traditional Internet context, this factor is about 1000 or 10,000. This means that, in grids, bandwidth demand from a single or a small set of source-sink pair can easily reach the scale of hundreds of Mbit/s to even several Gbit/s. Such giant tasks introduce a relatively low multiplexing level, while consuming a large portion of bandwidth in underlying networks.

On the other hand, TCP-based protocols are chosen by grid applications to exchange the data between end points because of their current availability in operating systems. Such distributed transport protocols are designed to statistically share available bandwidth among flows in a "fair" way. It is known that this core-stateless approach performs well unless total demand approaches full capacity of bottleneck link, which is relatively rare to happen in Internet [1]. For example, an OC12 link (622 Mbit/s) can concurrently support hundreds to thousands of flows from DSL lines (around 2 Mbit/s). In low multiplexing context, the usage of multi streams helps in increasing the multiplexing degree and in improving the throughput. This explains why tools like GridFTP, based on multiple and parallel streams, is widely used in grids to give users better performance. However, such artifacts do not provide any transfer delay guarantee. Indeed TCP/IP technology has not been designed to ensure predictable completion time of data transfers. Consequently, in grids, variation of incoming traffic load and dramatic extension of data transfer delay can easily cause computing tasks miss their deadlines [2]. Make things worse, as the per-flow product of bandwidth and latency increases, TCP becomes inefficient and prone to instability [3]. To avoid both user dissatisfaction and end point resource under-utilization, we believe that admission control and per-flow bandwidth allocation of bulk data transfers is necessary (and also practical) in high-end Grid networks, which are characterized by high QoS requirement and low aggregation factor. We propose a data mover service to carry out giant transfer tasks (with volume higher than several Gbyte) in specified time intervals.

* Corresponding author.
E-mail addresses: ssoudan@ens-lyon.fr (S. Soudan), chenbinb@comp.nus.edu.sg (B.B. Chen), Pascale.Primet@inria.fr (P. Vicat-Blanc Primet).

0167-739X/$ – see front matter © 2008 Elsevier B.V. All rights reserved.
doi:10.1016/j.future.2008.06.006

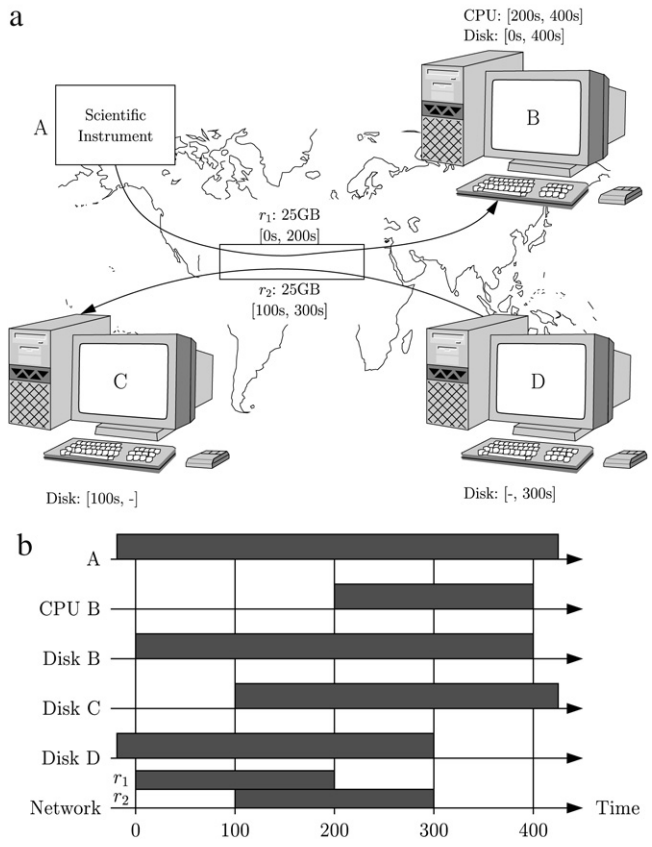

Fig. 1. Bulk data transfer scheduling example.

For example, as shown in Fig. 1, 25 GB data produced in site *A* needs to be moved to site *B* for processing. The CPU and disk resources in site *B* have been reserved in advance from 200 s to 400 s, and from 0 s to 400 s, respectively. If transfer can begin only when disk in site *B* is allocated, and there is no pipeline between transfer and computing services, the bulk data transfer task r_1 needs to move 25 GB data from site *A* to site *B* in the time interval [0 s, 200 s], to fully use the CPU resources. As another example, there is a 25 GB file stored in site *D* where the lease time will expire in 300 s, owner of the data reserves new storage spaces in site *C*, where the lease begins from 100 s. The bulk data transfer task r_2, thus, requires to move 25 GB data from site *D* to site *C* in the time interval of [100 s, 300 s].

Bandwidth reservation has been studied extensively for real-time applications [4], which are often approximately modeled as reserving a fixed amount of bandwidth from a given start time. In comparison, bulk data transfer tasks are specified in terms of *volume* and *active window* (from start time to deadline). With flexibility at its best, a bulk data transfer task can start from any time after its start time, use any and even time variant bandwidth value, as long as it is completed before its deadline. The underlying principle is to exploit this time flexibility to minimize congestion by shifting traffic from peak time to off-peak time.

The rest of paper is organized as follows. Section 2.1 presents the Bulk Data Transfer Scheduling Problem and a brief description of the BDTS service architecture. Section 3 examines the problem of end-host bandwidth profile enforcement which is a key component for a large scale deployment of this approach. Several solutions are discussed. The evaluation methodology and the experimental results of mechanisms available in the Linux kernel are given in Section 4. Related works are summarized in Section 5 before we conclude with Section 6.

2. Bulk data transfer scheduling problem

2.1. Problem formulation

Let R represent a set of data transfer requests over a network $G(V, E)$ consisting of node set V and edge set E, with edge capacity $\mu(e) : E \to \mathbf{R}^+ - \{0\}$, where $\mathbf{R}^+$ is the set of non-negative real numbers. Formally, if

- v_r is the *volume* of the data to transfer,
- $\omega_r = [\eta_r, \psi_r]$ is the *time window* (from *start time* η_r to *deadline* ψ_r),
- $|\omega_r| = \psi_r - \eta_r$ is the length of time window,
- $\Phi_r = \{\phi_r^1, \phi_r^2, \ldots, \phi_r^{|\Phi_r|}\}$ is the set of paths connecting source s_r and destination d_r,
- $r = (v_r, \omega_r, \Phi_r)$ represents a request,
- $\Omega = \bigcup_{r \in R} \omega_r$ is the union of all requests' time windows,
- $\lambda_\phi(t) : \omega_{r(\phi)} \to \mathbf{R}^+$, is the bandwidth allocation profile of a path, which specifies the amount of bandwidth reserved in every link $e \in \phi$ for path ϕ at time t,
- $f_e(t) = \frac{\lambda_e(t)}{\mu(e)}$, is link e's *congestion factor* at time t, and
- $f_{\omega e} = \max_{t \in \omega} f_e(t)$, denotes the *congestion factor* over time interval ω

then, the Bulk Data Transfer Scheduling problem, *BDTS(R,G)*, is to select a *bandwidth allocation profile* of a request r, $\lambda_r(t) : \omega_r \to \mathbf{R}^+$, for each request $r \in R$, so that the overall network congestion factor is minimized. We define the *network congestion factor* over ω as $f_{\omega G} = \max_{e \in E} f_{\omega e}$.

It has been shown [5] that the optimal network congestion factor does not change if we restrict the solution space to the practical multi-interval scheduling schemes, which only use bandwidth allocation profile taking the form of a step function. A n-piece step function $\lambda(t)$ defined over time window $\omega = [\eta, \psi]$ can be represented as:

$$\lambda(t) = a_0 + a_1 h_\omega^{b_1}(t) + a_2 h_\omega^{b_2}(t) + \cdots + a_{n-1} h_\omega^{b_{n-1}}(t) \tag{1}$$

where:

$$h_\omega^b(t) = \begin{cases} 0 & t \in [\eta, b) \\ 1 & t \in [b, \psi] \end{cases} \tag{2}$$

is a revised Heaviside step function (unistep function). For $i = 1, 2, \ldots, n-1$, b_i is the ith non-continuous point in (η, ψ), and $a_i = \lambda_{t \to b_i +}(t) - \lambda_{t \to b_i -}(t)$. $a_0 = \lambda(\eta)$.

A multi-interval scheduling scheme divides the active window of each job into multiple intervals, and reserves a constant bandwidth value (including zero) independently in each of them. The bulk data scheduling problem can be formulated as a linear programming problem [2] defined as follow:

$$\begin{aligned} &\textit{minimize} : f_{\Omega G} \qquad (3)\\ &s.t. \quad \sum_{\phi \in \Phi_r} \int_{\eta_r}^{\psi_r} \lambda_\phi(t) dt = v_r, \quad \forall r \in R; \\ &\qquad \sum_{\phi \in \Phi_e \cap \Phi(t)} \lambda_\phi(t) \le f_{\Omega G} * \mu(e), \quad \forall e \in E, \forall t \in \Omega; \\ &\qquad \lambda_\phi(t) : \omega_{r(\phi)} \to \mathbf{R}^+, \quad \forall \phi \in \Phi. \end{aligned}$$

The first constraint in formula (3) is the volume demand requirement. The integral of a request's bandwidth allocation profile, which is the sum of the integrals over all of its paths, is equal to its volume. The second one is the capacity constraint, which bounds the sum of the profiles of all active paths passing through a link. The third one gives the constraint on solution space. *BDTS* can be solved in polynomial time as a *Maximum Concurrent Flow Problem (MCFP)*, and the number of intervals used by each job's bandwidth reservation profile is upper bounded in the optimal solution attained as demonstrated in [2].