

# Uniform Job Scheduling Model for Distributed Processing Environment with Distributed Storage

Lukáš Hejtmánek\* and Petr Holub\*†

\*Faculty of Informatics and †Institute of Computer Science,  
Masaryk University Brno, Botanická 68a, 602 00 Brno, Czech Republic  
e-mail: xhejtman@mail.muni.cz, hopet@ics.muni.cz

## ABSTRACT

In this report, we describe components, algorithms and mathematical details for the scheduling jobs with respect to distributed both computing capacity and storage capacity. The work was designed for distributed video processing in Grid environments described in [5]. The video processing features important advantage of having almost exactly uniform job size and thus the scheduling algorithm may belong to *PO* class under certain assumptions discussed here. Furthermore we provide in-depth discussion of requirements on network traffic prediction services that need to be used to optimize data location with respect to available processing capacity and vice versa.

## Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>Use Cases and Scenarios</b>	<b>2</b>
<b>3</b>	<b>Variables, Conventions, and Comments</b>	<b>3</b>
<b>4</b>	<b>Components of the Model</b>	<b>4</b>
4.1	CTE – Completion Time Estimation . . . . .	4
4.2	Available bandwidth estimate . . . . .	5
4.3	Proximity Function . . . . .	7
4.4	Prefetch Evaluation . . . . .	8
4.5	Upload Optimizations . . . . .	8
<b>5</b>	<b>Scheduling algorithms</b>	<b>8</b>
5.1	Processor scheduling . . . . .	9
5.2	Storage scheduling problem, 1-to-1 model . . . . .	10
5.3	Storage scheduling problem, 1-to-n model . . . . .	11
<b>6</b>	<b>Conclusions</b>	<b>12</b>

# 1 Introduction

The work described in this report was primarily motivated by a need for efficient job scheduling across geographically distributed computing cluster infrastructure and distributed storage systems for distributed processing of large data sets. Such scheduling system must take into account not only the processing power of each computing node (which is not uniform as often understood by traditional scheduling algorithms), but also estimated end-to-end network throughput between the location of the data in the distributed storage system and the processing nodes. Furthermore, the primary application for this model – video processing system called Distributed Encoding Environment (DEE) described in more detail in [5] – has interesting property of generating computational jobs with approximately uniform size which is very advantageous, because it allowed designing scheduling mechanism that belongs to *PO* class shown below.

The pilot application DEE uses MetaCenter PC clusters [7] with deployed PBS Pro [8] system for simple job scheduling and submission and distributed storage infrastructure called Distributed Data Storage (DiDaS) [4, 3] based on Internet Backplane Protocol [2].

The report is organized as follows: in Sect. 2 we define several use cases this work is aimed to, Sect. 3 describes notation and conventions used in this document, Sect. 4 introduces basic components of the model, and Sect. 5 explains models under several different assumptions and discusses their complexity for each case.

## 2 Use Cases and Scenarios

There is a number of scenarios that can be covered by the approach described in this report. The following list includes the ones we consider the most important:

- *Selection of best hosts to perform the processing:* The “best” host doesn’t need to be the fastest one in terms of available processing power. Actually, it is the one on which the calculation finishes in shortest time. To select the best node for the processing, we need to sort hosts according to estimated completion time.
- *Prefetch decision support.* Some evaluation criteria need to be provided to decide whether data prefetch is appropriate or not. The minimal condition is that the prefetch must accelerate the processing, i. e. it must decrease overall processing time.
- *Upload distribution support.* If the data processing is to happen in short enough future so that we can predict which machines will be used, it is convenient to upload the data into the distributed storage with respect to available processing infrastructure.

### 3 Variables, Conventions, and Comments

There is number of variables used in this report and most of them are described in this section. Some variables are also provided with deeper explanation where appropriate.

$t$  time

$t_0$  now

$p$  processing node

$P$  set of processing nodes

$d$  the depot in which the data to be processed are stored

$D_u$  set of depots scheduled/used for actually accessing the data to be processed in task  $u$

$D$  set of depots that store data to be processed (all depots unless indicated otherwise)

$u$  (type of) processing task

$U$  set of processing tasks (all the tasks have the same length)

$l_u$  length of processing task  $u$ ; units [Mb]

$t_p^{\text{sched\_free}}$  information from job scheduler in what time the processor  $p$  will be available; units [s]

This information can be theoretically obtained from most of current advanced schedulers. However there are a few issues that make it partially theoretical functionality only:

- existence of priority jobs in scheduling systems (the priority job can delay availability of the processor)
- most users don't bother to specify expected run-time of their jobs thus defaulting to maximum run-time available in specifying queue
- bad quality of code of this functionality in well-known and used scheduling systems (e.g. PBS [8])

Nevertheless there is considerable effort in current Grid computing to make estimates of job run-times [6] and thus we assume this functionality available in the near future.

$s_{p,u}$  processing performance of processor  $p$  on (type of) task  $u$ ; units [Mb . s<sup>-1</sup>]

We assume that the processing performance of the processor is constant in time and the processor is either available (free) or unavailable (busy) for the purpose of job scheduling. When some algorithms assume uniform tasks, we denote  $s_p$  processing performance of processor  $p$ .

$b_{p,D}(t)$  download capacity (bandwidth) from depot set  $D$  to processor  $p$  in time  $t$ ; units [Mb . s<sup>-1</sup>]

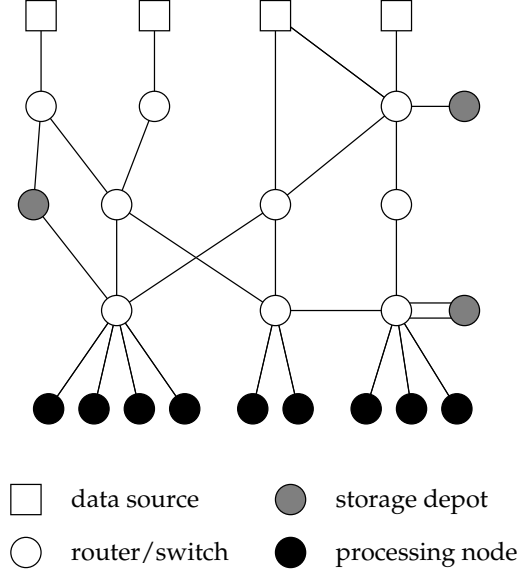


Figure 1: General model overview.

## 4 Components of the Model

There are two basic components of the model: *Completion Time Estimate (CTE)* used for finding the “best” host for data processing, and *Network Traffic Prediction Service (NTPS)* for prediction of the available end-to-end bandwidth between data storage depot and processor. Furthermore, some auxiliary functionality like proximity functions, prefetch decision support, and upload optimizations are provided in this section.

### 4.1 CTE – Completion Time Estimation

In general, CTE can be obtained by solving the following equation for the job  $u$  given location of the data in depots  $D_u$  and using processor  $p$  and resulting data of length  $l_u^{out}$  are uploaded into depot set  $D'_u$

$$\int_{t_p^{sched\_free}}^{CTE_d(p, D_u, u)} \min\{s_{p,u}, b_{D_u,p}(t)\} dt = l_u \quad (1)$$

$$\int_{t_p^{sched\_free}}^{CTE_u(p, D_u, u)} b_{p, D'_u}(t) dt = l_u^{out} \quad (2)$$

$$CTE(p, D_u, u) = \max\{CTE_d(p, D_u, u), CTE_u(p, D_u, u)\} \quad (3)$$

The  $CTE_d(p, D_u, u)$  is estimated completion time of download and processing phase and the  $CTE_u(p, D_u, u)$  is estimated completion time of upload phase.

This model also presumes that the uploading into the storage infrastructure takes place in parallel with downloading otherwise the lower bound in (2) needs to be modified accordingly (e. g. when uploading happens just after the processing finishes, the lower bound would be  $\text{CTE}_d(p, D_u, u)$ ).

If we assume that  $b_{p,D_u}(t)$  is constant in interval  $\langle t_p^{\text{sched\_free}}, \text{CTE}(p, D_u, u) \rangle$  (which can be justified e. g. since job duration is less than time resolution of network traffic prediction service), we can use simplified model.

$$\text{CTE}(p, D_u, u) = t_p^{\text{sched\_free}} + \max \left\{ \frac{l_u}{\min\{s_{p,u}, b_{D_u,p}(t_p^{\text{sched\_free}})\}}, \frac{l_u^{\text{out}}}{b_{p,D'_u}} \right\} \quad (4)$$

If we again assume uploading phase just after processing finishes, the (4) transforms into

$$\text{CTE}(p, D_u, u) = t_p^{\text{sched\_free}} + \frac{l_u}{\min\{s_{p,u}, b_{D_u,p}(t_p^{\text{sched\_free}})\}} + \frac{l_u^{\text{out}}}{b_{p,D'_u}} \quad (5)$$

To simplify the model even further, we can assume that the uploading into the infrastructure is not the bottleneck since  $l_u^{\text{out}} \ll l_u$  (which is typical for video processing applications from raw video format to compressed formats) while  $b_{D_u,p} \approx b_{p,D'_u}$ , or that the uploading phase takes negligible time only even if the uploading occurs after the processing. Thus we obtain formula that will be used further on in this paper for sake of simplicity

$$\text{CTE}(p, D_u, u) = t_p^{\text{sched\_free}} + \frac{l_u}{\min\{s_{p,u}, b_{D_u,p}(t_p^{\text{sched\_free}})\}} \quad (6)$$

In case that the presumption with neglecting uploading phase is not valid, the model and the resulting algorithm can be easily extended to support it.

Such function allows us to find the most suitable processors for processing. To avoid synchronous overloading of processing infrastructure, we suggest to use one of the two well-known approaches:

- randomize set of processing nodes and pre-select some subset
- pre-select the subset manually

For the given subset we calculate CTE estimates and launch a greedy scheduling algorithm starting with processor with lowest CTE.

## 4.2 Available bandwidth estimate

Let's assume we have some kind of Network Traffic Prediction Service (NTPS) that provides us with estimate of available network throughput between node  $A$  and node  $B$  in time  $t$ :  $\text{NTPS}(A, B, t)$ . For receiving realistic estimate of available TCP bandwidth, at least following parameters need to be evaluated: minimum line capacity on the path, round-trip time (RTT), and packet loss rate as all of these are important for performance of TCP that is underlying our applications. There are several possible models of NTPS with different interactions with our job scheduling model as shown below.

The main difficulty arises when the traffic generated by "our" application has regular patterns in its nature and thus it is included as a part of NTPS

prediction itself. In such scenario we need to differentiate between predicted traffic generated by “our” application and predicted background traffic. Moving from most complex to simpler models, we will show interactions with our scheduling system for each of them.

### NTPS Model #1

Let’s assume model with following properties:

1. the NTPS can predict cumulative available TCP bandwidth in N:1 fashion when N host are sending data to single host in parallel,
2. “our” application tells the NTPS which traffic in NTPS measurements has been generated by it to identify it,
3. the NTPS service can provide prediction of “background” traffic for “our” application by subtracting eliminating predicted traffic of “our” application from overall predicted traffic,
4. the NTPS performs in-advance bandwidth allocations in time for scheduled jobs and project these allocations into available bandwidth predictions,
5. the NTPS can compare reserved vs. actual traffic by “our” application and it can utilize it to keep statistic information (which can be e. g. automatically used if application always overestimates bandwidth needed in its allocation),
6. the NTPS can estimate the available bandwidth in end-to-end way; that means it can decrease available bandwidth correspondingly if the bottleneck is either in storage depot itself of processing node itself.

Under such conditions what we need from the prediction service is total bandwidth available between processor  $p$  and depot set  $D$ ,

$$b_{D,p}(t) = \text{NTPS}(D, p, t) \quad (7)$$

The application then allocates following bandwidth per processor  $b_{p,d}^{\text{sched}}(t)$  with NTPS service unless the total reserved bandwidth is larger than  $s_{p,u}$ . It must take previous allocations from different depots into account as well

$$b_{d,p}^{\text{sched}}(t) = \min \left\{ s_{p,u}, b_{d,p}(t), \max \left\{ \left( s_{p,u} - \sum_{d'} b_{p,d'}^{\text{sched}}(t) \right), 0 \right\} \right\} \quad (8)$$

where  $d'$  ranges over depots that are already scheduled to be used. The depots from  $D$  will be added unless  $b_{p,d}^{\text{sched}}(t) = 0$  or no other depot in  $D$  is available.

### NTPS Model #2

If the NTPS prediction is unable to perform prediction in N:1 fashion (relaxing condition 1), it is near to impossible to use multiple depots to feed one processor as it requires knowledge of network topology. Thus the formulas above become simplified

$$b_{D,p}(t) = \text{NTPS}(d, p, t) \quad \text{where } D = \{d\}, \quad (9)$$

$$b_{d,P}^{\text{sched}}(t) = \min \{s_{p,u}, b_{d,p}(t)\} \quad (10)$$

### NTPS Model #3

If we assume the same behavior as above (Model #2) with exception of unavailable bandwidth allocations (relaxing conditions 1 and 4), the model gets more complicated again even if we use single depot per processor only.

$$b_{D,p}(t) = \text{NTPS}(d, p, t) - \sum_{p'} b_{d,p'}^{\text{sched}}(t) \quad \text{where } D = \{d\}, \quad (11)$$

where  $p'$  goes over all processes that share some link with previously scheduled processing (thus omitting this term when creating the first estimate)— $(d, p)$  and  $(d, p')$  share at least one link.

### NTPS Model #4

If the network traffic forecasting service is capable of including “our jobs” into its estimate but it is unable to isolate “our” traffic from its predictions, the calculation becomes:

$$\text{NTPS}(D, p, t) > 0 \quad \text{or} \quad \text{NTPS}(d, p, t) > 0 \quad (12)$$

as we are watching whether there is still some spare bandwidth available to say whether the congestion (including “our traffic”) is imminent or not.

### NTPS Model #5

If we are sure the NTPS doesn’t include “our” traffic in its prediction (e. g. since “our” traffic is neither regular nor predictable), the formula becomes

$$b_{D,p}(t) = \text{NTPS}(D, p, t) \quad \text{or} \quad b_{D,p}(t) = \text{NTPS}(d, p, t) \quad \text{where } D = \{d\}, \quad (13)$$

## 4.3 Proximity Function

The auxiliary proximity function allows us to have assessment of “closeness” of processors and storage depots in a static time average fashion. In similar way to NTPS function, the proximity functions must take into account maximum achievable end-to-end throughput depending in link capacities, RTTs, and packet loss rates and also bandwidth inside both end nodes. Proximity functions can be prototyped as follows:

- PX( $p$ )    ... returns vector of depots close to  $p$  (in non-increasing order)
- PX<sup>inv</sup>( $d$ )    ... returns vector of processors close to single depot  $d$   
(in non-increasing order)
- PX<sup>inv</sup>( $D$ )    ... returns vector of processors close to depot set  $D$   
(in non-increasing order)

#### 4.4 Prefetch Evaluation

First, it makes no sense to perform the prefetch if the processing power is the bottleneck, so the prefetch makes sense only if

$$s_{p,u} > \frac{\int_{t_p^{\text{sched\_free}}}^{\text{CTE}_d(p,D,u)} b_{D,p}(t) dt}{\text{CTE}_d(p,D,u) - t_p^{\text{sched\_free}}} = \hat{b}_{D,p} \quad (14)$$

It is meaningful to perform prefetch from depots  $D$  to depots  $D'$  if following condition is met:

$$t_p^{\text{sched\_free}} + \frac{l_u}{\min\{s_{p,u}, b_{D,p}(t_p^{\text{sched\_free}})\}} > t' + \frac{l_u}{\min\{s_{p,u}, b_{D',p}(t')\}} \quad (15)$$

where

$$t' = \max\{t_p^{\text{sched\_free}}, t_0 + \Delta t_{D \rightarrow D'}^{\text{prefetch}}\} \quad (16)$$

It is also necessary to find out whether there is some available depot which is "closer" than current ones. Minimal condition to attempt to do prefetch is that for any  $p \in P$

$$\exists d' \in \left\{ \left( \bigcup_i \text{PX}_i(p) \right) - D \right\}. \quad b_{d',p}(t) > 0. \quad (17)$$

where  $\text{PX}_i(p)$  is the  $i$ -th element of vector  $\text{PX}(p)$ . If we want to maintain number of copies of the data and just "flow" the data in the storage infrastructure, the condition looks as follows

$$\exists d' \in \left\{ \left( \bigcup_i \text{PX}_i(p) \right) - D \right\} \wedge \exists d \in D. \quad b_{d',p}(t) > b_{d,p}. \quad (18)$$

Simplified version for single storage of data

$$\text{PX}_0(p) \neq d \quad (19)$$

#### 4.5 Upload Optimizations

If we know at the time of uploading data from source nodes into data storage depots that there is some pool of processors  $P$  we want to use and assuming that the storage infrastructure can perform auto-replication and prefetching, we can upload data to following set of depots

$$\bigcup_{p \in P} \text{PX}_0(p) \quad (20)$$

### 5 Scheduling algorithms

Our model consists of two stages. First, processor scheduling algorithm assigns tasks to the processors and in the second storage scheduling that assigns tasks to depots. We use an abstraction that we can see these two parts independently. Practical implementation of course needs to schedule processors

and depots at once while depots are sharing interconnecting network lines. We are assuming two models of storage scheduling: first one is 1-to-1 where one task data is stored only in a single depot. Second 1-to-n where one task data is replicated to  $n$  depots that can be accessed simultaneously. Last but not least, it is important to keep in mind that our model assumes uniform tasks only, as discussed in Section 1.

## 5.1 Processor scheduling

We are not using online algorithm and we use an abstraction that all the tasks are known at time of scheduling. For real online algorithm, this might not hold. Furthermore, for purpose of this algorithm we don't care about the available network capacity between storage depots and processing nodes and the only measure of speed is  $s_{p,u}$ , which is denoted as  $s_p$  because of uniform job size.

**Input:** set of processors  $P$ , set of tasks  $U$ , task length  $l$ , speed of each processor  $s_p$ .

**Output:** sets  $U_p$  that contain tasks assigned to processor  $p$ , and for  $\forall u \in U$  scheduled time to start the task  $t_u$  is computed.

**Goal:** minimize.

**Measure:** maximum processor running time.

Common processor scheduling problem, which takes uniform processors and tasks of different sizes, belongs to  $NPO$  class. In this case we have uniform task size and processors with different speeds.

Let  $t_p^{sched-free}$  be the time when processor  $p$  is free (meaning there is no task scheduled to processor  $p$  at time  $t_p^{sched-free}$ ). We are using greedy algorithm (see Figure 2) for assigning tasks to processors. It is easy to see that complexity of algorithm is  $O(|P||U|)$ .

```

1 foreach  $p \in P$  do
2    $U_p := empty$ ;
3    $t_p^{sched-free} := 0$ ;
4 od
5 foreach  $u \in U$  do
6    $p := \forall p_1 \in P \ t_{p_1}^{sched-free} + \frac{l_u}{s_{p_1}} \geq t_p^{sched-free} + \frac{l_u}{s_p}$ ;
7    $U_p := U_p \cup \{u\}$ ;
8    $t_u := t_p^{sched-free}$ ;
9    $t_p^{sched-free} := t_p^{sched-free} + \frac{l_u}{s_p}$ ;
10 od

```

Figure 2: Greedy algorithm for processor scheduling

**Theorem 1:** Processor scheduling belongs to  $PO$  class.

**Proof:** We need to show that greedy algorithm (Fig. 2) returns optimum solution to prove that processor scheduling belongs to  $PO$  class. Since all the

tasks are uniform and no task precedence is required, we can see that any permutation of tasks inside  $U_p$  does not result in better or worse solution. Moreover, let  $u_1 \in U_{p_1}$  and  $u_2 \in U_{p_2}$  for two processors  $p_1 \neq p_2$ , we can see that  $\{U_{p_1} - u_1\} \cup \{u_2\}$ ,  $\{U_{p_2} - u_2\} \cup \{u_1\}$  does not give better solution, because the tasks are of uniform size. Let  $u_1 \in U_{p_1}$  and  $U_{p_2}$  where  $p_1 \neq p_2$ , we show that  $\{U_{p_1} - u_1\}$ ,  $\{U_{p_2} \cup u_1\}$  does not give better solution. Since we can do any permutation of tasks in any  $U_p$ , let  $u_1$  be the task whose  $t_{u_1}$  is maximal in  $U_{p_1}$ , i. e. it is the last scheduled task in  $U_{p_1}$ . Let  $u_2 \in U_{p_2}$  whose  $t_{u_2}$  is maximal in  $U_{p_2}$ .

- In the case of  $t_{u_2} + \frac{l_{u_2}}{s_{p_2}} + \frac{l_{u_1}}{s_{p_2}} > t_{u_1} + \frac{l_{u_1}}{s_{p_1}}$  worse solution is found.
- In the case of  $t_{u_2} + \frac{l_{u_2}}{s_{p_2}} + \frac{l_{u_1}}{s_{p_2}} < t_{u_1} + \frac{l_{u_1}}{s_{p_1}}$  a better solution is found, but we show that this is impossible.

Holding for  $\forall p' \in P$ .  $t_{p'}^{sched-free} + \frac{l_{u_1}}{s_{p'}} \geq t_{u_1} + \frac{l_{u_1}}{s_{p_1}}$  (line 6 in Fig. 2), we substitute variables:  $t_{u_2} + \frac{l_{u_2}}{s_{p_2}} + \frac{l_{u_1}}{s_{p_2}} \geq t_{u_1} + \frac{l_{u_1}}{s_{p_1}}$ . But that is a contradiction with  $t_{u_2} + \frac{l_{u_2}}{s_{p_2}} + \frac{l_{u_1}}{s_{p_2}} < t_{u_1} + \frac{l_{u_1}}{s_{p_1}}$ .  $\square$

## 5.2 Storage scheduling problem, 1-to-1 model

**Input:** set of all depots  $D$ , set of processors  $P$ , set of tasks  $U$ , speed of each processor  $s_p$ , transfer speed between processor and depot  $b_{d,p}(t)$ , for  $\forall p \in P$  scheduled  $U_p$ , and for  $\forall u \in U$  scheduled  $t_u$ .

**Output:** sets  $P_d$  that contain tasks assigned to depot  $d$ .

**Goal:** maximize.

**Measure:**  $\sum_{u \in U} f(b_{d,p}(t_u) - s_p)$

where  $d \in D$  is such that  $u \in P_d$ ,  $p \in P$  is such that  $u \in U_p$ , and

$$f(x) = \begin{cases} 1 & x \geq 0 \\ 0 & x < 0 \end{cases}$$

**Theorem 2:** The 1-to-1 storage scheduling problem is *NPO*-complete.

**Proof:** We use Karp reduction to Bin-Packing problem. Let  $I = \{a_1, a_2, \dots, a_n\}$  be the finite set of rational numbers with  $a_i \in (0, 1]$  for  $i = 1, \dots, n$ , we search minimal partition  $\{B_1, B_2, \dots, B_k\}$  of  $I$  such that  $\sum_{a_i \in B_j} a_i \leq 1$  for  $j = 1, \dots, k$ . Let  $s_{p_1} = a_1, s_{p_2} = a_2, \dots, s_{p_n} = a_n$  for 1-to-1 storage scheduling problem. Let processors and depots are interconnected via the complete graph. Let  $\forall d \in D : \sum_{p \in P} b_{d,p}(t) \leq 1$  for any time  $t$ . We search minimum set of depots  $D$  so that  $\sum_{u \in U} f(b_{d,p}(t_u) - s_p)$  where  $d \in D$  is such that  $u \in P_d, p \in P$

is such that  $u \in U_p$ , and  $f(x) = \begin{cases} 1 & x \geq 0 \\ 0 & x < 0 \end{cases}$  is maximal. That means we run algorithm for 1-to-1 storage scheduling up to  $k+1$ -times. Depots corresponds to partition  $\{B_1, B_2, \dots, B_k\}$ .  $\square$

We use an approximative algorithm First Fit Decreasing that is used for Bin-packing problem. The First Fit Decreasing is  $\frac{3}{2}$ -approximative scheme [1].

```

1 foreach  $d \in D$  do
2    $P_d := \text{empty}$ ;
3 od
4 foreach  $p \in P$  do
5    $t_p^{\text{sched-free}} := \text{PBS}(p)$ ;
6    $\text{ProcTime}(p) := \frac{t_u}{\min\{s_p, b_{D,p}(t_p^{\text{sched-free}})\}}$ ;
7 od
8 foreach  $u \in U$  do
9    $p : \forall p_1 \in P, t_{p_1}^{\text{sched-free}} + \text{ProcTime}(p_1) \geq t_p^{\text{sched-free}} + \text{ProcTime}(p)$ ;
10   $d : \forall d_1 \in D, b_{d_1,p}(t_p^{\text{sched-free}}) \leq b_{d,p}(t_p^{\text{sched-free}})$ ;
11   $\text{sched\_depot}(u, d)$ ; /*  $P_d := P_d \cup \{u\}$  */
12   $\text{sched\_job}(p, u)$ ; /*  $t_p^{\text{sched-free}} := t_p^{\text{sched-free}} + \text{ProcTime}(p)$  */
13 od

```

Figure 3: 1-to-1 task scheduling

The algorithm for processor and task scheduling is in Figure 3. The function  $\text{sched\_job}(p, u)$  tells the cluster's resources scheduling system (e. g. PBS) to allocate processor time starting at  $t_p^{\text{sched-free}}$  and to mark particular processor busy. The function  $\text{sched\_depot}(p, d)$  changes network and depot conditions  $b_{d,p}(t_p^{\text{sched-free}})$  in the way that data transfer from depot  $d$  to processor  $p$  will utilize network and depot capacity.

### 5.3 Storage scheduling problem, 1-to-n model

**Input:** set of depots  $D$ , set of processors  $P$ , set of tasks  $U$ , speed of each processor  $s_p$ , transfer speed between processor and depot  $b_{d,p}(t)$ , for  $\forall p \in P$  scheduled  $U_p$ , and for  $\forall u \in U$  scheduled  $t_u$ .

**Output:** sets  $P_d$  that contain tasks assigned to depot  $d$ .

**Goal:** maximize.

**Measure:**  $\sum_{u \in U_p} f(\sum_{d \in D_u} b_{d,p}(t_u) - s_p)$

$$\text{where } D_u = \{d \mid u \in P_d\}, p \in P, \text{ and } f(x) = \begin{cases} 1 & x \geq 0 \\ 0 & x < 0 \end{cases}$$

Algorithm in Figure 4 is a modified version of algorithm for 1-to-1 task scheduling that uses replicas of the data on different data storage depots to find optimum solution. Transfers to processors can be done from multiple sources. Please note, that we work in the complete graph to achieve  $PO$ -class complexity and thus line 10 in Fig. 4 can work with  $b_d$  separately instead of  $b_D$ , since there is no shared link and thus the data flows are independent and thus the  $b_d$  is additive.

**Theorem 3:** The 1-to-n storage scheduling belongs to  $PO$  class if and only if depots and processors are interconnected via the complete graph.

```

1 foreach  $u \in U$  do
2    $D_u := \text{empty}$ ;
3 od
4 foreach  $d \in D$  do
5    $P_d := \text{empty}$ ;
6 od
7 foreach  $p \in P$  do
8    $t_p^{\text{sched-free}} := PBS(p)$ ;
9    $ProcTime(p) := \frac{l_u}{\min\{s_p, b_{D,p}(t_p^{\text{sched-free}})\}}$ ;
10 od
11 foreach  $u \in U$  do
12    $p : \forall p_1 \in P. t_{p_1}^{\text{sched-free}} + ProcTime(p_1) \geq t_p^{\text{sched-free}} + ProcTime(p)$ ;
13   while  $(s_p > b_{D_u,p} \wedge \exists \text{ free depot})$  do
14      $d : \forall d_1 \in D. b_{d_1,p}(t_p^{\text{sched-free}}) \leq b_{d,p}(t_p^{\text{sched-free}})$ ;
15      $\text{sched\_depot}(u, d)$ ; /*  $P_d := P_d \cup \{u\}, D_u := D_u \cup \{u\}$  */
16   od
17    $\text{sched\_job}(p, u)$ ; /*  $t_p^{\text{sched-free}} := t_p^{\text{sched-free}} + ProcTime(p)$  */
18 od

```

Figure 4: 1-to-n task scheduling

**Proof:** We need to show that greedy algorithm returns optimum solution to prove that 1-to-n storage scheduling belongs to  $PO$  class. Indeed, using replicas allows us to utilize all the depots to their maximum, this means that no better solution can be found. In the case of non complete graphs some network conditions can prevent utilizing some depots to the maximum extent when First Fit Decreasing algorithm is used. (i.e. the second fastest processor is connected only to the fastest depot while the first fastest processor is connected to all depots and is fast enough to utilize the fastest depot to its maximum then the second fastest processor has no access to free depot.)  $\square$

**Theorem 4:** The 1-to-n storage scheduling is  $NPO$ -complete if depots and processors are interconnected via common graph.

**Proof:** The proof uses the same reduction as proof for the theorem 2 while network conditions restrict using of replicas. So there can exist depot that is not utilized to maximum while there is a processor that is neither utilized to the maximum extent and using more replicas is not more efficient.  $\square$

## 6 Conclusions

We have proposed scheduling mechanisms for data processing with both distributed storage and processing capacities. In case of uniform jobs and 1-to-n task to depot scheduling, the algorithm turns out to be  $PO$ -class problem. We have also discussed all the components necessary for such model. While the most of the components are readily available or will be available in the near future, we still don't see any network traffic prediction service under active

development, that could satisfy our requirements discussed in depth in Sect. 4. This issue should be further investigated.

## References

- [1] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi: *Complexity and Approximation—Combinatorial Optimization Problems and Their Approximability Properties*. Springer Verlag, November 1999, ISBN 3-540-65431-3. <http://www.nada.kth.se/~viggo/approxbook/>.
- [2] M. Beck, T. Moore, and J. S. Planck: “An end-to-end approach to globally scalable network storage.” In *SIGCOMM’02*, 2002. <http://loci.cs.utk.edu/ibp/files/pdf/SIGCOMM02p1783-beck.pdf>.
- [3] L. Hejtmánek: “Distributed data storage based on web access and ibp infrastructure.” In *Data Processing and Storage Networking: Towards Grid Computing, Technical Proceedings. Third IFIP-TC6 Networking Conference*, May 2004. <http://www.ece.ntua.gr/networking2004/>.
- [4] L. Hejtmánek and P. Holub: *IBP deployment tests and integration with DiDaS project*. Technical Report 20/2003, CESNET, 2003. <http://www.cesnet.cz/doc/techzpravy/2003/ibpdidas/>.
- [5] P. Holub and L. Hejtmánek: “Distributed encoding environment based on grids and ibp infrastructure.” In *TERENA Networking Conference 2004*, page 10, June 2004, ISBN 90-77559-04-3. <http://www.terena.nl/library/tnc2004-proceedings/>.
- [6] M. Sgaravatto, P. Andreetto, S. Borgia, A. Dorigo, A. Gianelle, M. Mordacchini, L. Zangrando, S. Andreozzi, V. Ciaschini, C. D. Giusto, F. Giacomini, V. Medici, E. Ronchieri, V. Venturi, G. Avellino, S. Beco, A. Maraschini, F. Pacini, A. Guarise, G. Patania, D. Kouril, A. Krenek, L. Matyska, M. Mulac, J. Pospsil, M. R. Z. Salvat, J. Sitera, J. Skrabal, M. Vocu, V. Martelli, M. Mezzadri, F. Prelz, D. Rebatto, S. Monforte, and M. Pappalardo: *Practical approaches to grid workload and resource management in the egee project*, 2004. <http://egee-jral-wm.mi.infn.it/egee-jral-wm/documents/wms.pdf>.
- [7] *MetaCenter Project*, CESNET. <http://meta.cesnet.cz/>.
- [8] *Portable Batch System*. Altair Engineering, <http://www.pbspro.com/>.