

Data popularity and shortest-job-first scheduling of network transfers

Simone Lupetti, Dmitrii Zagorodnov

Department of Computer Science
University of Tromsø

E-mail: {simone, dmitrii}@cs.uit.no

Abstract

This paper presents a strategy for scheduling transfers of data items from a single source to a single destination. We are motivated by the scenarios in which an author with limited connectivity relies on a remote publishing server for sharing data items with readers. Our strategy is based on the Shortest Job First (SJF) algorithm with the computational cost expressed in terms of the transfer time and the number of readers waiting for an item. We consider cases with static and dynamic popularity of items. We prove that our strategy is optimal in the static case. For the dynamic case, we use simulation results to demonstrate that our strategy results in lower average waiting time when compared to an SJF algorithm that bases its decision only on the expected transfer time or only on the popularity of a data item.

1 Introduction

This paper presents a novel strategy for scheduling transfers of data items from a single source to a single destination. This strategy lowers the average waiting time for a group of readers waiting for the items to become accessible at the destination. It is suitable for scenarios in which an author with limited connectivity – insufficient bandwidth, underpowered server resources, or a long-latency link to its readers – relies on a remote publisher for sharing data items with the readers.

This work is motivated by the typical Web publishing infrastructures of today, in which the content is uploaded from an author’s host to a Web server and not directly to the readers. If the data items are large with respect to the upload link capacity – as in the case of sending high-resolution images or long video clips over a broadband connection or over a wireless link of a mobile device – the author may be interested in reducing the time the readers have to wait for the items of their choice to become available. Concrete examples of this are Web-based personal file sharing services

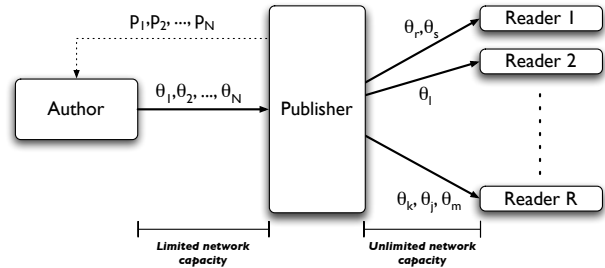


Figure 1. The modeled scenario: A set of items (θ_i) is being uploaded to the publisher. Some of them will be requested by the readers. The publisher informs the author about the popularity (p_i) of each item.

like .Mac [1] and Streamload [3] that allow users to share large files, as well as services specifically for hosting multimedia content. Content distribution networks, such as Akamai [15, 9] and Mirror Image Internet [2], in which large files are replicated on servers that are geographically close to users, can also be viewed as examples of publishers serving authors that have limited connectivity. If files are replicated “on demand,” e.g. in response to an increase in popularity, then minimizing the waiting time can be critical for maximizing performance.

Our strategy is a variation of the Shortest Job First (SJF) algorithm with tasks being data item transfers and with the computational cost for each task expressed in terms of both the expected transfer time for an item and its popularity. The scheduler can compute the former based on the size of the item; the latter comes from the publisher in the form of a count of the readers requesting the item. Multiple requests from a reader for the same item are treated as one.

We assume that for each run the author has a finite set of data items to upload. The publisher knows the set before the transfer begins (e.g. by receiving an index of all items) and is able to store all of them. Furthermore, we assume unlimited link capacity between the readers and the publisher, but limited link capacity between the publisher and

the author. Thus, in our model, a reader may request any item in the set as soon as the upload of the set from the author commences; the item becomes available to the reader as soon as it has been fully uploaded to the publisher. The time between the arrival of the first request for an item and the completion of its upload is *waiting time*. The item will remain on the publisher to service future requests promptly (with the waiting time of zero). This setup is illustrated in Fig. 1 with items shown as θ_i and their popularity as p_i .

These assumptions are realistic for the examples that motivated this work, in which the author-to-publisher upload process is the bottleneck (if it is not, there is no need for sophisticated scheduling). This bottleneck appears whenever the ratio of data size to link bandwidth is high. For example, transferring the contents of a 500-GB hard disk over Gigabit Ethernet and transferring a dozen videos, each 10 MB, over an ADSL connection (maximum 128 kbit/s upstream), both require over an hour. The transfer time is similar for a dozen high-resolution images, e.g. 6.5-MB “raw” files from an 8-megapixel camera, traversing a 3G wireless link. In the evaluation of our work, we do not consider link bandwidth explicitly, but instead we experiment with a wide range of item transfer times, which can be related to concrete scenarios, such as those listed above.

We present two versions of the proposed scheduling strategy. The first one, described in Sec. 2, is *static*, in the sense that the popularity of data items is fixed and known when the upload starts. We prove that the SJF scheduling based on the ratio of size to popularity results in the minimal average waiting time.

The second strategy, presented in Sec. 3, allows for requests to arrive during the upload. Using a simulator, we demonstrate that in this case basing SJF scheduling on the ratio of size to popularity often results in significantly lower average waiting time than basing scheduling on size or popularity alone. Finally, we considered the issue of preemption, i.e. interrupting a transfer to schedule a higher-priority item, and found that it lowers average waiting time only with small item sets (at most 6 items).

2 Static Case

In this case, we assume that the popularity of each item is fixed and known before the transfer starts. This means the scheduler only needs to make a decision once, at the beginning of the transfer. Replication of files from an overloaded server to remote content distribution servers is an example of such a scenario. Since the original server is responding to the growth in popularity of certain files, their access pattern can be used to make a static scheduling decision. The definitions and the proof in this section serve as a starting point for the next section’s discussion of dynamic scheduling.

Consider a set Θ of N tasks (the data items to be uploaded, in our case):

$$\Theta = (\theta_1, \theta_2, \dots, \theta_N). \quad (1)$$

If time starts at the moment when the set Θ is ready to be transferred to the publisher, and if f_i is the time when task θ_i has been uploaded, then the *average waiting time* (AWT) for all tasks of this set under a schedule σ is

$$AWT(\sigma) = \frac{1}{N} \sum_{0 < i \leq N} f_i(\sigma). \quad (2)$$

This definition of AWT expresses the average of the waiting times experienced by a reader waiting for every task from the beginning of the transfer. To account for items requested by more than one reader, we include the popularity p_i of item θ_i (i.e. the number of readers served by it) into the expression:

$$AWT(\sigma) \triangleq \frac{1}{M} \sum_{0 < i \leq N} p_i f_i(\sigma). \quad (3)$$

And instead of averaging over the total number of tasks N , we average over the total number of outstanding requests M :

$$M \triangleq \sum_{0 < i \leq N} p_i, \quad p_i > 0 \quad \forall i.$$

Notice that if $p_i = 1 \quad \forall i$, then (3) is equivalent to (2). The unrequested items ($p_i = 0$) are not included in the waiting time since nobody waits for them. We assume that their transfer is deferred until all requested items have been transferred. Each term $p_i f_i(\sigma)$ of (3) represents the cumulative waiting time for all the readers that requested the task θ_i .

2.1 Static Algorithm

It is well known that the basic SJF algorithm results in the minimal AWT of (2). (See, for instance, the discussion of shortest-processing time scheduling in Conway et al. [8].) This algorithm schedules the tasks according to their computational cost c_i (the size of a data item in our case) with the smaller tasks preceding the bigger tasks.

The basic SJF algorithm needs to be modified to accommodate (3):

Theorem 1. To minimize (3), a scheduling algorithm must schedule tasks in the order of increasing *equivalent computational cost*, defined as

$$c_i^* = \frac{c_i}{p_i}, \quad (4)$$

where c_i is the computational cost of the task θ_i and p_i is its popularity.

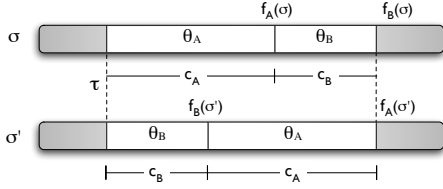


Figure 2. Two schedules in the static case: σ (optimal) and σ' , differing in the order of tasks θ_A and θ_B .

Proof 1. Consider two schedules, σ and σ' , identical but for the order of two adjacent tasks θ_A and θ_B , as shown in Fig. 2. We prove that if the optimal schedule prioritizes the costlier of the two tasks, then we get a contradiction. Assuming σ is the optimal schedule, from (3) we get

$$\frac{1}{M} \sum_{0 < i \leq N} p_i f_i(\sigma) \leq \frac{1}{M} \sum_{0 < i \leq N} p_i f_i(\sigma'). \quad (5)$$

Since the final times f_i of all tasks other than θ_A and θ_B is the same for both schedules (see Fig. 2), we define the quantity α as:

$$\alpha = \frac{1}{M} \sum_{\substack{0 < i \leq N \\ i \neq A, B}} p_i f_i(\sigma) = \frac{1}{M} \sum_{\substack{0 < i \leq N \\ i \neq A, B}} p_i f_i(\sigma').$$

Now (5) can be rewritten to highlight the differences between the two schedules as:

$$\begin{aligned} \frac{1}{M} \left(\alpha + f_A(\sigma) p_A + f_B(\sigma) p_B \right) \\ \leq \frac{1}{M} \left(\alpha + f_A(\sigma') p_A + f_B(\sigma') p_B \right). \end{aligned} \quad (6)$$

From Fig. 2 one can see that

$$\begin{aligned} f_A(\sigma) &= \tau + c_A & f_B(\sigma) &= \tau + c_A + c_B \\ f_B(\sigma') &= \tau + c_B & f_A(\sigma') &= \tau + c_B + c_A, \end{aligned}$$

where τ is the sum of all computational costs of the tasks before θ_A in the schedule σ and before θ_B in the schedule σ' . By substituting the previous expression of f_A and f_B in (6) and simplifying the common terms on both sides we obtain

$$c_A p_B \leq c_B p_A,$$

and, since $p_i > 0 \forall i$ (we only deal with items that are requested), we have

$$\frac{c_A}{p_A} \leq \frac{c_B}{p_B}.$$

Or, in other words,

$$c_A^* \leq c_B^*,$$

which contradicts our initial hypothesis that $c_A^* > c_B^*$. Therefore for σ to be the optimal schedule, $c_i^* \leq c_{i+1}^*, \forall i$ must hold. ■

To obtain the lowest AWT, tasks must be scheduled in the order of the increasing equivalent computational cost defined in (4). Essentially, the computational cost c_i of θ_i is prorated by the number p_i of users served by that task.

When the behavior of readers is unpredictable, one cannot assume that the popularity p_i of each data item is fixed and known when the transfer of items begins. A more general model would accommodate changes in the popularity of data over time $p_i(t)$ and would extend the definition of AWT to allow the requests to arrive at any time. Such a model is presented next.

3 Dynamic Case

When requests for a specific data item arrive dynamically, the waiting time for each reader depends both on the arrival time of the request and on the time when the item is completely uploaded at the publisher (f_i for item θ_i). Since the global number of requests M for all items is not known until all of them are transferred, we define the *number of requests* at time k as

$$M[k] = \sum_{0 < i \leq N} p_i[k],$$

where $p_i[k]$ is the number of requests for the data item θ_i at time k . The expression of the AWT in the dynamic case is then also dependent on k . In particular, the AWT for the whole set of data items is available only for $k \geq f_N$, where f_N is the finishing time of the last task. We then extend the expression of AWT to accommodate dynamic requests as

$$AWT[k] \triangleq \frac{1}{M[k]} \sum_{0 < i \leq N} S_i[k], \quad (7)$$

where

$$S_i[k] \triangleq \sum_{j=1}^{\min\{k, f_i\}} (p_i[j] - p_i[j-1])(f_i - j) \quad (8)$$

is the contribution at time k of task θ_i to the total AWT. In other words, this term represents the number of users that have requested the task θ_i and for how long each of them had already waited for it at time k . Each $S_i[k]$ is the product of two distinct parts:

$$(p_i[j] - p_i[j-1]),$$

which is the number of requests that arrived between $j-1$ and j for the task θ_i , and

$$(f_i - j),$$

which is how long that task had to wait, either until the current step k or until the finishing time f_i , whichever is earlier, as selected by $\min\{k, f_i\}$.

3.1 Dynamic algorithm

In Sec. 2 we augmented the standard SJF algorithm with a new notion of computational cost, so that the AWT of (3) is minimized. When requests arrive dynamically, however, no optimal algorithm can be derived because no strategy can minimize the AWT – as defined in (7) – without knowing the pattern of future requests $p_i[k]$.

For a sketch of a proof, consider a set of tasks made up of only two items θ_A and θ_B . The scheduler must choose at $k = 0$ whether to first transfer θ_A or θ_B . However, at that time it is not possible to estimate the contributions of the two tasks to the total waiting time since that depends on the pattern of future requests $p_i[k]$ (with $0 < k \leq f_N$), which is not known until $k = f_N$. Since it is not possible to choose between two tasks, it is also not possible to do it with more than two.

Nevertheless, it is still possible to extend the notion of the equivalent computational cost from Sec. 2 to the dynamic case, and compare it to other approaches using simulations. We define the dynamic equivalent computational cost as

$$c_i^*[k] = \frac{c_i}{p_i[k]}, \quad (9)$$

where c_i is the expected transfer time for the item θ_i (like for the static case) and $p_i[k]$ is its popularity at time k .

3.2 Algorithm evaluation

Here we present the results of simulating the behavior of the SJF that uses (9) as the notion of cost. We call this approach the *size/popularity* algorithm. We compare it to the SJF that uses as cost just the item size or just the item popularity. We refer to the former as the *size-only* algorithm and to the latter as the *popularity-only* algorithm.

We implemented two queues on the author: a *priority queue* for the items that have received at least one request and a *pre-fetching queue* for all other items of the set. The size-only algorithm does not consider requests at all and uses only the pre-fetching queue. At the beginning of the transfer, all items are in the latter queue but as soon as requests start to arrive, the requested items are moved from the pre-fetching queue to the priority one. The items in the pre-fetching queue are scheduled for transfer using the size-only algorithm. Items in transfer from the pre-fetching queue are preempted by the ones in the priority queue.

3.2.1 Experimental setup

We measured the AWT, as defined in (7), in a simulation of a transfer of a set of 100 items with a fixed mean item size. Requests for items were scattered uniformly throughout the transfer period (Poisson arrivals) and each scenario was evaluated between 0.01 and 10000 requests per average transfer time with a resolution of 5 sample points/decade.

Although the mean item size was the same in all experiments, we varied the distribution of item sizes. We experimented with sizes uniformly distributed between several pairs of values, including a pair of identical values, which resulted in the special case of a constant size for all items. We also experimented with the Pareto distribution using different values of α and β to control the shape and the location of the tail.

Similarly, we varied the probability of an item being requested, i.e. the distribution of popularity among items. We tried both uniform and normal distributions, with different variance values to control the extent to which certain items dominate in popularity. We experimented with all possible combinations of size and popularity distributions, trying different configurations of each, but we present here only the most representative results.

Each presented AWT is the mean of 10 runs, each using a different request pattern $p_i[k]$ with the same item set Θ , and 90% confidence intervals shown for determining whether the difference between algorithms is significant. In all experiments we consider the expected transfer time to be directly proportional to the item's size. Therefore, our model assumes that the upstream bandwidth of the author does not vary during the transfer.

3.2.2 Discussion

We found that the proposed size/popularity algorithm always gives at least equal and often better AWT than the size-only and the popularity-only algorithms. The magnitude of the performance improvement, however, depends on the request arrival ratio.

Fig. 3 shows the scenario in which all items have the same size and the uniformly distributed popularity (i.e. they all have the same probability of being requested). Here, size/popularity performs better than size-only, while it offers the same performance as popularity-only. For high volumes of requests – more than 3000 requests/average transfer time – all three algorithms behave equally well. The reason for this is that the differences in popularity become negligible for high request rates since the requests are uniformly distributed over the set of items.

For very low request rates, the difference between size/popularity and size-only decreases because when no requests arrive for a specific item before it is transferred, its

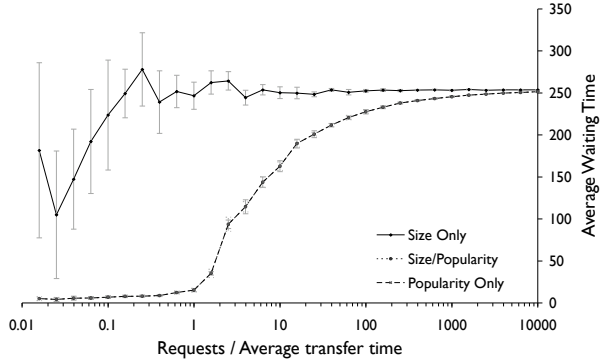


Figure 3. AWT of three SJF algorithms given constant item size (15) and uniform distribution of requests.

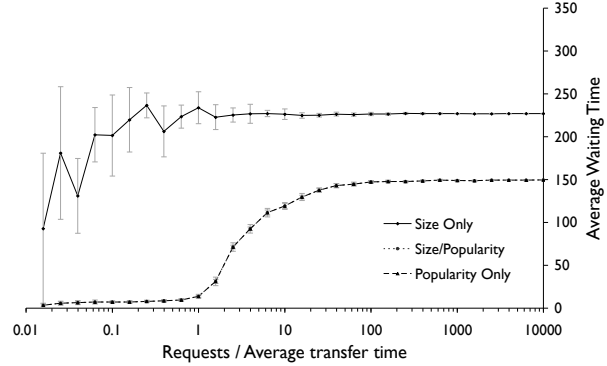


Figure 5. AWT of three SJF algorithms given constant item size (15) and Gaussian ($\mu = 50, \sigma = 25$) distribution of requests.

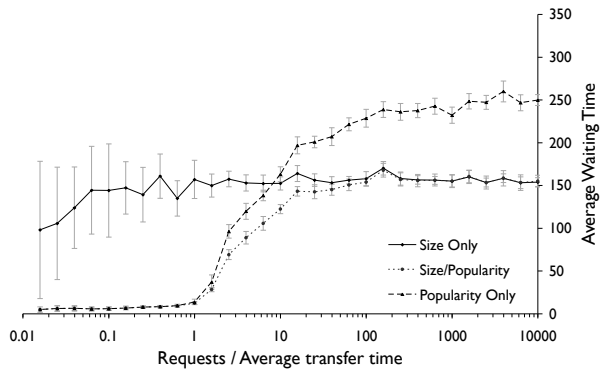


Figure 4. AWT of three SJF algorithms with uniformly distributed item sizes (1–29) and uniform distribution of requests.

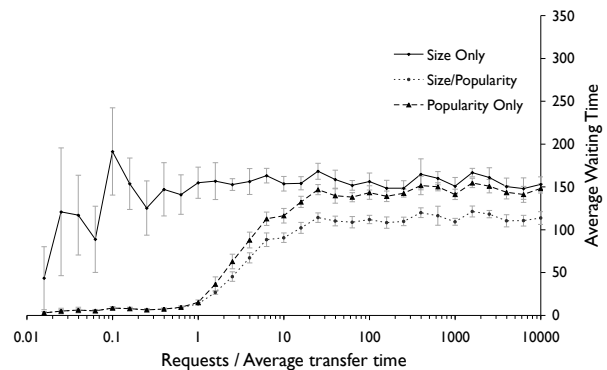


Figure 6. AWT of three SJF algorithms with uniformly distributed item sizes (1–29) and Gaussian ($\mu = 50, \sigma = 25$) distribution of requests.

contribution to the AWT is zero, as can be seen in (8). In this range (request rate ≤ 1), two behaviors are noteworthy:

First, the size-only algorithm has large confidence intervals because the AWT varies a lot from run to run. This is because when few requests arrive, the AWT is determined on the basis of few contributions. If a request arrives for a large item (that will be scheduled late in the run), then the AWT will be high. If instead a request arrives for a small item (that may have been already transferred) the resulting AWT will be close or equal to zero.

Second, when the arrival rate of requests is equal to the transfer time for an item, the expected AWT for the two strategies that take into account the popularity of items is exactly equal to the average transfer time (15 in all our experiments). This is because when a request arrives, it is likely to be the only one to be served. The requested item is therefore immediately scheduled for transfer, making the AWT only equal to the time necessary to transfer the item.

When item size varies, both the size/popularity and the size-only algorithm result in lower AWT at high request rates, as shown in Fig. 4. The popularity-only algorithm, which ignores the size, results in the same values of AWT as in Fig. 3. In this setting, with uniformly distributed item sizes, the proposed size/popularity algorithm always performs better than the other two in the range of 1-200 requests/average transfer time.

Next, we explored the scenario in which some items are more popular than others by using a Gaussian distribution of popularity. Fig. 5 shows the results of that simulation with a constant item size. The two algorithms that take into consideration item popularity perform identically, achieving lower AWT than with uniform request distribution (Fig. 3).

Results of combining varying item sizes and varying popularity – the most realistic configuration so far – are shown in Fig. 6. In this case the proposed algorithm outperforms the other two when more than 1 request per average

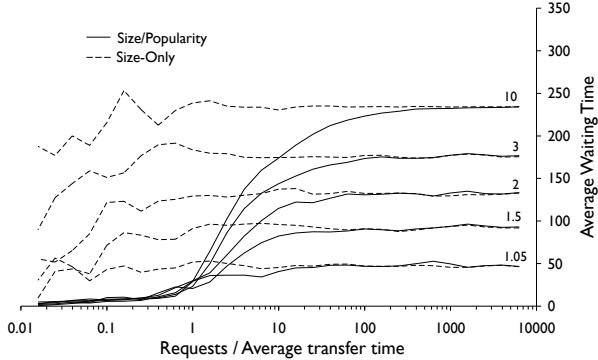


Figure 7. AWT of the *size-only* and *size/popularity* SJF algorithms with Pareto distributed item sizes at different values of the shape factor α , from 1.05 to 10.

transfer time arrives.

In many large collections of documents, including files transported over the Internet using TCP, the size follows “long tailed” distributions [19]. To study the SJF algorithms under those circumstances, we experimented with a Pareto distribution and varied the shape factor α , which determines where the long tail starts. To maintain the constant mean item size, we truncated the tail of the Pareto distribution at 30 times the mean item size.

Figure 7 shows the AWT with different shape factors for the *size-only* and *size/popularity* algorithms (the performance of the popularity-only algorithm is independent from the item size distribution). We observed an increase in the AWT as α increased (i.e. as the probability distribution curve got steeper). For $\alpha \geq 10$ the variance among sizes was small enough ($\mu \leq 2.2 \cdot 10^{-3}$) that the results resemble the scenario with a constant item size (Fig. 3). With small α , on the other hand, both *size-only* and *size/popularity* algorithms take advantage of the large difference in the item sizes and result in low AWT. As before, *size/popularity* performs better at low request rates.

3.2.3 Preemption

So far, we have considered algorithms in which item transfers proceed uninterrupted. To understand whether preempting a transfer can lead to a lower AWT, we simulated the SJF algorithms above with the computational cost defined as

$$c_i^*[k] = \frac{c_i^{left}[k]}{p_i[k]},$$

where $c_i^{left}[k]$ denotes the remaining computational time of task θ_i at time k . This cost was recomputed not only when each transfer finished, but also when each request arrived.

If the cost was lower for an item different from the one currently being transferred, the transfer was preempted and the item with the lowest cost was sent instead.

Surprisingly, we found no significant differences between preemptive and non-preemptive versions. We attribute this to the relatively high number of items (100) in the set Θ . Simulations with small item sets showed that the preemptive algorithm gives a lower AWT (at most 10% improvement, with a set of two items) until the set size reaches six. Beyond that, preemptive and non-preemptive versions perform identically for every request arrival rate. However, if the assumption of Poisson arrival of requests does not hold, preemption can produce benefits with larger item sets.

4 Related Work

The scheduling problem has been studied extensively and a large body of literature is available on the subject [4, 5, 7].

From the theoretical point of view, the general problem of scheduling N processes on M machines has been demonstrated as NP-hard for N and M larger than two [11, 14]. However, non-optimizing solutions for many common scenarios have been developed. As early as 1976, Panwalkar et al. [18] surveyed over 100 different scheduling heuristics.

Some of the solutions are specifically for file transfer scheduling [12, 13]. This paper reexamines the file scheduling problem in a novel setting – a file publishing infrastructure that reports item popularity – and presents a thorough analysis of both static and dynamic cases.

During the last few years, perhaps due to the emergence of the content distribution networks (CDNs), file transfer scheduling research gained in popularity. Several teams worked on the problem of lowering the transfer time for a set of nodes involved in peer-to-peer exchange of files over a mesh topology [6, 10, 16]. In contrast, we schedule transfers of data items over a single link and aim to lower the average waiting time. In our model, the readers, which are equivalent to the clients in a CDN, are not known to the author.

Outside the domain of file scheduling, shortest-job-first processing techniques have recently been shown to minimize the length of queues in Internet routers and, thus, alleviate congestion [17].

5 Conclusions

We have presented a variation of the Shortest Job First algorithm for scheduling transfers of a set of data items from an author with limited connectivity to a publisher with no connectivity limitations. The variation consisted of redefining

the computational cost for each task in terms of the ratio of the expected transfer time c_i for an item i to its popularity p_i . Essentially, our strategy favors the most popular and the smallest items: they become available as soon as possible, thus lowering the waiting time for the readers.

The formula for the new computational cost c_i^* is

$$c_i^*[k] = \frac{c_i[k]}{p_i[k]}.$$

We proved that scheduling data items by placing the least costly ones first minimizes the Average Waiting Time (AWT), as defined in (2), when p_i does not change with time and the scheduling decision is made once, at the beginning of the transfer (the static case).

We then extended this result to derive an algorithm that lowers the AWT, as defined in (7), when the popularity $p_i[k]$ changes over time and scheduling decisions are made upon completion of each transfer (the dynamic case). We showed how this algorithm performs better overall than a simpler SJF that uses as computational cost only the item size or item popularity. Furthermore, we found that unless the item set is very small (six or fewer items), preempting transfers does not help performance.

In the future, we plan to enlarge the scope of our investigation to scenarios with non-Poisson request arrival patterns (most notably, the “flash crowd” effect), as well as to the more complex item distribution topologies.

6 Acknowledgments

This work has been generously supported by the Research Council of Norway by means of the Arctic Bean project (IKT 2010, project number 146986/431). The authors would like to thank the anonymous reviewers for their helpful comments.

References

- [1] .Mac. <http://www.apple.com/dotmac/>.
- [2] Mirror Image Internet. <http://www.mirror-image.com/>.
- [3] Streamload. <http://www.streamload.com/>.
- [4] K. Baker. *Introduction to sequencing and scheduling*. Addison-Wesley, 1974.
- [5] P. Brucker. *Scheduling Algorithms*. Springer, 4th edition, 2004.
- [6] L. Cherkasova and J. Lee. FastReplica: Efficient large file distribution within content delivery networks. In *Proceedings of the 4th USENIX Symposium on Internet Technologies and Systems*, 2003.
- [7] E. G. Coffman, J. L. Bruno, and R. L. Graham. *Computer and job-shop scheduling theory*. Addison-Wesley, 1976.
- [8] R. W. Conway, W. L. Maxwell, and L. W. Miller. *Theory of Scheduling*. Addison-Wesley, 1967.
- [9] J. Dille, B. Maggs, J. Parikh, H. Prokop, R. Sitaraman, and B. Weihl. Globally distributed content delivery. *IEEE Internet Computing*, 6(5):50–58, Sept. 2002.
- [10] J. E. G. Coffman, M. R. Garey, D. S. Johnson, and A. S. LaPaugh. Scheduling file transfers in a distributed network. In *PODC '83: Proceedings of the second annual ACM symposium on Principles of distributed computing*, pages 254–266, New York, NY, USA, 1983.
- [11] S. French. *Sequencing and Scheduling: An Introduction to the Mathematics of the JobShop*. John-Wiley & Sons, 1982.
- [12] M. N. Garofalakis and Y. E. Ioannidis. Scheduling issues in multimedia query optimization. *ACM Comput. Surv.*, 27(4):590–592, 1995.
- [13] A. Goel, M. R. Henzinger, S. Plotkin, and E. Tardos. Scheduling data transfers in a network and the set scheduling problem. In *STOC '99: Proceedings of the thirty-first annual ACM symposium on Theory of computing*, pages 189–197, New York, NY, USA, 1999.
- [14] N. G. Hall and C. Sriskandarajah. A survey of machine scheduling problems with blocking and no-wait in process. *Operations Research*, 44(3):510–525, May–Jun 1996.
- [15] D. Karger, E. Lehman, T. Leighton, R. Panigrahy, M. Levine, and D. Lewin. Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the World Wide Web. In *Proceedings of the 29th ACM Symposium on the Theory of Computing (STOC)*, pages 654–663, May 1997.
- [16] D. Kosti, A. Rodriguez, J. Albrecht, and A. Vahdat. Bullet: high bandwidth data dissemination using an overlay mesh. In *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 282–297, New York, NY, USA, 2003.
- [17] D. McNickle and R. G. Addie. Comparing protocols for differential service in the Internet. In *Proceedings of IEEE TENCON 2005*. IEEE, December 2005.
- [18] S. S. Panwalkar and W. Iskander. A survey of scheduling rules. *Operations Research*, 1(25):45–61, Jan–Feb 1976.
- [19] K. Park, G. Kim, and M. E. Crovella. On the relationship between file sizes, transport protocols, and self-similar network traffic. Technical Report BU-CS-96-016, Boston University, August 1996.