An Experimental Study of *k*-Splittable Scheduling for DNS-Based Traffic Allocation

Amit Agarwal[⋆], Tarun Agarwal[⋆], Sumit Chopra[⋆⋆], Anja Feldmann[⋆⋆⋆], Nils Kammenhuber[⋆⋆⋆], Piotr Krysta[†], and Berthold Vöcking[‡]

Topic 3: Scheduling and Load Balancing

Abstract. The Internet domain name system (DNS) uses rotation of address lists to perform load distribution among replicated servers. We model this kind of load balancing mechanism in form of a set of request streams with different rates that should be mapped to a set of servers. Rotating a list of length k corresponds to breaking streams into k equally sized pieces. We compare this and other strategies of how to break the streams into a bounded number of pieces and how to map these pieces to the servers.

One of the strategies that we study computes an *optimal k-splittable allocation* using a scheduling algorithm that breaks streams into at most $k \geq 2$ pieces of possibly different size and maps these pieces to the servers in such a way that the maximum load over all servers is minimized. Our experimental study is done using the network simulator SSFNet. We study the average and maximum delay experienced by HTTP requests for various traffic allocation policies and traffic patterns. Our simulations show that splitting data streams can reduce the maximum as well as the average latency of HTTP requests significantly. This improvement can be observed even if streams are simply broken into k equally sized pieces that are mapped randomly to the servers. Using allocations computed by machine scheduling algorithms, we observe further significant improvements.

1 Introduction

The Internet's Domain Name System (DNS) is responsible for resolving host names like "`www.tum.de`" into IP addresses. This service is also being used to perform load distribution among distributed Web servers. Busy sites are replicated over multiple servers, with each server running on a different end system having its own IP address. Thus a set of IP addresses is associated with one canonical hostname contained in the DNS database. When clients make a DNS query for such a host name, the server returns the entire set of addresses,

[⋆] IIT Delhi, India. This work was done while the author was visiting the MPI in Saarbrücken.
[⋆⋆] Department of Computer Science, Hansraj College, University of Delhi, India. This work was done while this author was visiting the MPI in Saarbrücken.
[⋆⋆⋆] Department of Computer Science, Technische Universität München, Germany.
[†] Max-Planck-Institut für Informatik, Saarbrücken, Germany.
[‡] Department of Computer Science, Universität Dortmund, Germany.

thereby rotating the order of addresses within each reply. Because a client typically sends its HTTP request to the server listed first, the traffic is distributed among all replicated servers (cf. [7]).

One can view the requests that are directed to the same URL as traffic streams. The rotation of address lists basically splits these streams into equally sized pieces. Suppose the request streams are formed by a sufficiently large number of clients. Then the arrivals of requests can be described by a stochastic process. In the scenario that we consider there are n streams and m identical servers. Let λ_j denote the rate of stream $j \in [n]$, i. e., the expected number of requests in some specified time interval. Under this assumption, rotating a list of k servers corresponds to splitting stream j into k substreams each of which having rate λ_j/k. The following slightly more sophisticated stochastic splitting policy would possibly achieve a better load balancing. Suppose, the DNS attaches a vector $p_1^j, \dots, p_k^j$ with $\sum_i p_i^j = 1$ to the list of each stream j. In this way, every individual request in stream j can be directed to the ith server on this list with probability p_i^j. This policy breaks Poisson stream j into k Poisson streams of rates $p_1^j\lambda_j, \dots, p_k^j\lambda_j$, respectively.

The focus of this paper is not on how exactly to implement such strategies within the DNS implementation. Instead, our goal is to study the impacts of different allocation strategies a few steps ahead of current DNS implementations, including calculating an optimal k-split allocation. The optimal k-split allocation is computed using a variant of a recently presented algorithm for the k-splittable machine scheduling problem. Suppose a set of jobs (data streams) $[n] = \{1, \dots, n\}$ of sizes $\lambda_1, \dots, \lambda_n$ need to be scheduled on a set of identical machines (web servers) $[m] = \{1, \dots, m\}$. Every job can be broken into at most k pieces. The goal is to map job pieces to machines so that the makespan, i. e. the maximum load over all machines, is minimized.

Scheduling with bounded splittability was introduced by Shachnai and Tamir [9]. They prove that the problem is NP-hard and present approximation algorithms. Krysta et al. [6] presented an exact algorithm for the k-splittable machine scheduling problem with running time $O(m^{m+m/k} + n)$. Thus the problem can be solved in polynomial time for any fixed number of machines. It is well known that the classical unsplittable scheduling problem is NP-hard for 2 machines, so the result from [6] proves that bounded splittability reduces the problem's complexity for a fixed number of machines.

We remark that the algorithms in [6, 9] not only work for identical but also for machines of different speeds. In this paper, we focus on the special case of identical machines. Note that the above and all following bounds on the running time refer to algorithms that compute feasible allocations for any given upper bound on the makespan. These algorithms, in turn, can be used to compute the optimal makespan by applying binary search techniques.

When implementing the algorithm from [6], we learned that the exponential term in the running time let us only compute optimal assignments for relatively small numbers of machines. For some randomly generated instances with less than 40 machines the algorithm did not terminate in reasonable time. It was obvious, however, that, in the case of identical machines, one could exploit symmetries to speed up the algorithm. Clearly, because of the NP-hardness of the problem, the best one can hope for is to soften the exponential influence of the number of the machines. The theoretical contribution of this paper is an upper bound of $O\left((2m)^{m/(k-1)+1}(m/k)+n\right)$ on the running time of an exact algorithm for the k-splittable scheduling problem on identical machines. Observe that this result yields the first complete tradeoff on the influence of different degrees of splittability on the running time ranging from polynomial time in the case of $k = \Omega(m)$ to exponential time when $k = O(1)$.

In Section 2, we present this improved algorithm. Furthermore, we give some experimental results for different input distributions showing that the improved algorithm can be used to compute optimal solutions for several hundred machines very efficiently. In Section 3, we present our comparative study of different traffic allocation strategies applied to a simple network topology in which we measure average and maximum delays of HTTP requests. For a brief summary of these results, we refer the reader to the Conclusions presented in Section 4.

2 An improved algorithm for *k*-splittable scheduling

In this section, we present an improved algorithm for k-splittable scheduling on identical machines. We refer the reader to [6] for the original algorithm.

Let $x_{i,j}$ denote the fraction of job j that goes to machine i, let k_j be the current splittability of j. Of course, at the beginning we have $\forall j : k_j = k$. The *bulkiness* of a job j is defined as $\lambda_j/(k_j - 1)$. Assuming that a value z for the makespan (maximum load) is given, we look for an algorithm that either computes an assignment satisfying $\max_{i\in[m]} \sum_{j\in[n]} \lambda_j x_{ij} \leq z$ or returns that there is no z. Given such an algorithm, the original optimization problem can be solved in a polynomial number of iterations by applying binary search techniques over the rational numbers [5, 8]. The binary search works despite the possibility of irrational splits because the value of the optimal solution can be proved to be rational [6].

2.1 The improvement

We change the algorithm only in a small aspect: we reduce search space size by exploiting symmetries. Considering all machines with the same remaining capacity as belonging to one equivalence class, there is no need to backtrack over machines in the same class. In the case of identical machines, all *empty machines* (remaining capacity $= z$) fall into the same equivalence class. In contrast, *partially filled machines* (remaining capacity in $(0,z)$) typically build equivalence classes of size one. For simplicity of the implementation, we thus only exploit the symmetry among empty machines. When picking the jobs in the order of their bulkiness, we distinguish three cases (j denotes the job's index):

1) If $k_j \geq 2$ then we saturate one of the machines, selecting a machine from the class of the empty machines fi rst and then trying all machines with reduced capacity.
2) If $k_j = 1$ then we put the remaining piece job j to a machine i with $c_i \geq \lambda_j$, fi rst trying all partially fi lled machines and then one of the empty machines.
3) If $\frac{\lambda_j}{k_j - 1} \leq \min_{i \in [m]}(c_i)$ then the McNaughton phase is initiated.

Obviously, our modifi cation does not affect the algorithm's correctness. However, one might object that exploiting symmetries cannot help too much as the assignment of jobs destroys these symmetries rapidly. Nevertheless, the following theorem states that the running time is improved dramatically.

Theorem 1. *The running time of the improved algorithm for the k-splittable traffic allocation problem is* $O\left((2m)^{m/(k-1)+1}(m/k) + n\right)$*, for every* $k \in \{2,\ldots,m\}$.

The proof of this theorem is provided in [11]. Observe that the theorem yields a complete tradeoff on the influence of the different degrees of splittability on the running time. In particular, the following result was not known before.

Corollary 1. *For any constant* $c > 0$ *and* $k \geq \max\{2, \frac{m}{c}\}$*, k-splittable scheduling on identical machines is solvable in polynomial time.*

We implemented both the original algorithm and its modifi cation to compare their running times. An evaluation showed that also in practice, the modifi ed algorithm performs signifi cantly better than the original one. For details, we refer the reader to [11].

3 Evaluation of server load balancing simulations

To check how well our algorithms performs in the server farm load balancing scenario, we compared their results against three heuristics, i. e., *random allocation* (assign each job to a random machine), *random k-split allocation* (split

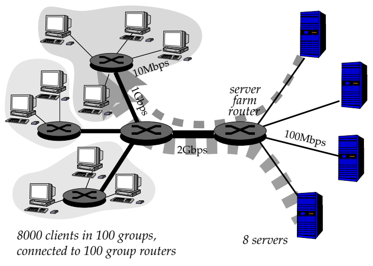

Fig. 1. Topology of simulated network.

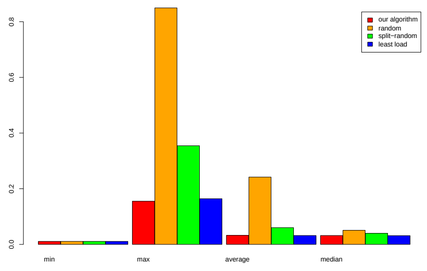

Fig. 2. Experienced response delays

each job into *k* equal parts, assign each part to a random machine) and *least-load* (“LL” — sort the jobs by their rates, split each job into *k* equal parts, assign each part to the machine with the smallest load at the time of scheduling).

For various *k* values, we created 10 random job assignment problems with uniform machine capacities. This was done for uniform, equally-distributed, exponentially distributed and Pareto-distributed job sizes, the latter for various α values. From the solutions gained by the algorithm (or the heuristics, respectively), we created network setups where each machine is represented by a web server and each job is represented by a group of uniform clients. The number of clients in a group reflects the workload of that job. Job assignment is modelled in a probabilistic way: Each group is given a probability vector that defi nes the probabilities under which the clients will issue their requests to what server. The network setups then were simulated using *SSFNet* [10].

We started using relatively simple network setups and experimented with various parameters, e. g. the link delays and random distributions of the object sizes, and slowly moved on to complex scenarios. We refer the reader to [11] for further details.

Among other evaluations, we calculated the delays between a request being issued and the fi rst byte of the response arriving at the client. Finally, we used a realistic web workload model for the client’s inter-request times, sizes of the objects being served, etc., which was derived from [1] and [3]. The network topology used is shown in Fig. 1.

An evaluation of some simulation runs with this setup is depicted in Fig. 2. We see that our algorithm clearly wins over the randomized schemes. It also wins over the simple LL heuristics, however the gain is rather narrow. Although Fig. 2 being an example, we note that these are tendencies also present in other simulation runs [11].

4 Conclusions

We studied delay and bandwidths experienced by HTTP requests for various traffi c allocation policies and traffi c patterns. Our simulation results show that splitting data streams into a small number of pieces can reduce the maximum as well as the average latency of HTTP requests signifi cantly.

Besides to random allocations, we devised and studied allocation strategies that allocate pieces of jobs to machines with the objective to minimize the maximum load over all servers. These strategies lead to a clear improvement in the maximum as well as the average latency of HTTP requests. It turned out, however, that the differences between the latencies obtained by the simple LL heuristic and the latencies obtained by an optimal k-splittable allocation are small.

There are several questions left open by our analysis. Certainly, some more test runs with the very large network setup would be interesting. It also could make sense to make the simulation environment even more realistic by studying even more complex network topologies. A second interesting topic are dynamic allocation schemes that use splittability to realize a smooth rather than an abrupt adaptation to dynamically changing traffi c patterns. These are topics that we plan to investigate in future work.

References

1. P. Barford, M. Crovella. *Web Server Workload Characterization: The Search for Invariants.* Proceedings of the ACM Sigmetrics Conference on Measurement and Modeling of Computer Systems 1996, pp. 126–137
2. M. Crovella, A. Bestavros. *Self-Similarity in World Wide Web Traffi c: Evidence and Possible Causes.* IEEE/ACM Transactions on Networking'96.
3. A. Feldmann, A. Gilbert, P. Huang and W. Willinger. *Dynamics of IP traffi c: a study of the role of variability and the impact of control*. Proceedings of SIGCOMM'99.
4. M. R. Garey and D. S. Johnson. *Computers and intractability: a guide to the theory of NP-completeness.* Freeman, 1979.
5. St. Kwek and K. Mehlhorn. *Optimal search for rationals.* Information Processing Letters, 86:23–26, 2003.
6. P. Krysta, P. Sanders and B. Vöcking. *Scheduling and Traffi c Allocation for Tasks with Bounded Splittability.* Proc. of the 28th International Symposium on Mathematical Foundations of Computer Science (MFCS), to appear, 2003.
7. J. F. Kurose, K. W. Ross. *Computer Networking: a top-down approach featuring the Internet.* Addison-Wesley, 2001.
8. C. H. Papadimitriou. *Effi cient search for rationals.* Information Processing Letters, 8:1–4, 1979.
9. H. Shachnai and T. Tamir. *Multiprocessor Scheduling with Machine Allotment and Parallelism Constraints.* Algorithmica, 32(4): 651–678, 2002.
10. Scalable Simulation Framework Research Network (SSFNet). *http://www.ssfnet.org/*
11. A. Agarwal, T. Agarwal, S. Chopra, A. Feldmann, N. Kammenhuber, P. Krysta, B. Vöcking. *An Experimental Study of k-Splittable Scheduling for DNS-Based Traffi c Allocation.* Technical Report TUM-I0304, Technische Universität München, 2003.