Brief Announcement: Full Reversal Routing as a Linear Dynamical System [*]

Bernadette Charron-Bost
Ecole polytechnique
charron@lix.polytechnique.fr

Matthias Függer[†]
TU Wien
fuegger@ecs.tuwien.ac.at

Jennifer L. Welch[‡]
Texas A&M University
welch@cse.tamu.edu

Josef Widder[§]
Texas A&M University
widder@cse.tamu.edu

ABSTRACT

Although substantial analysis has been done on the Full Reversal (FR) routing algorithm since its introduction by Gafni and Bertsekas in 1981, a complete understanding of its functioning—especially its time complexity—has been missing until now. In this paper, we derive the first exact formula for the time complexity of FR: given any (acyclic) graph the formula provides the exact time complexity of any node in terms of some simple properties of the graph. Our major technical insight is to describe executions of FR as a dynamical system, and to observe that this system is linear in the min-plus algebra.

As a consequence of the insight provided by the new formula, we are able to prove that FR is time-efficient when executed on tree networks. This result exposes an unstable aspect of the time complexity of FR that has not previously been reported. Finally, our results for FR are instrumental in providing an exact formula for the time complexity of a generalization of FR, as we show in a companion paper that the generalization can be reduced to FR.

Categories and Subject Descriptors: C.2.4 [Computer-Communication Networks]: Distributed Systems; F.2.m [Analysis of Algorithms and Problem Complexity]: Misc ellaneous; G.2.2 [Discrete Mathematics]: Graph Theory.

General Terms: Algorithms, Theory.

Keywords: routing, link reversal, time complexity, linear dynamical systems, min-plus algebra.

Introduction. Link reversal is a versatile algorithm design paradigm, originally proposed by Gafni and Bertsekas [6] for the problem of routing to a destination node in a wireless network subject to link failures. It has been used in solutions to resource allocation, distributed queuing, and various problems in mobile ad-hoc networks as routing, mutual exclusion, and leader election.

Link reversal is a way to change the orientation of links in a directed graph in order to accomplish some goal. In this paper, we focus on the problem of routing to a destination node; the goal is to ensure that, starting from some initial directed graph, ultimately every node in the graph has a (directed) path to the destination. Nodes that are *sinks* (that is, have no outgoing links) reverse the direction of a subset of their incident links. Different link reversal algorithms correspond to different choices of which incident links to reverse. The original paper [6] focused on two schemes, one called *Full Reversal (FR)*, in which all links incident on a sink are reversed, and the other called *Partial Reversal (PR)*, in which, roughly speaking, sinks only reverse those incident links that have not been reversed since the last time this node was a sink. The mechanism employed in [6] was to assign to each node a unique value called a *height*, to consider the link between two nodes as directed from the node with larger height to the node with smaller, and to reverse the direction of a link by increasing the height of a sink.

Surprisingly, there was no systematic study of the complexity of these algorithms until that of Busch *et al.* [2, 3]. In these papers, the authors considered the height-based implementations of FR and PR, and analyzed the *work* complexity, which is the total number of reversals done by all the nodes. For FR, an exact formula was derived for the work complexity of any node in any graph, implying that the total work complexity in the worst case is at most quadratic in the number of nodes; a family of graphs was presented to show that this worst-case quadratic bound is tight. Similar results were given for PR with asymptotically tight bounds.

The other natural complexity measure for link reversal algorithms is *time*, which is the number of iterations required until termination in "greedy" executions [1], where, in each iteration, all sinks take steps. Greedy executions are those with the highest possible parallelism. Clearly, global work complexity is the number of iterations in completely sequential executions, and so is at least equal to global time complexity. For both FR and PR, this implies a quadratic upper bound on global time complexity. Concerning time complexity, Busch *et al.* [2, 3] only obtained limited results: for each of FR and PR, they described a family of graphs on which the algorithm achieves quadratic global time com-

[*]A longer version of this paper is available as *TU Wien, Institut für Technische Informatik, Research Report 7/2011.*

[†]Supported in part by the FATAL project (P21694) of the Austrian Science Fund (FWF)

[‡]Supported in part by NSF grant 0964696

[§]Supported in part by NSF grant 0964696

SPAA'11, June 4–6, 2011, San Jose, California, USA.
ACM 978-1-4503-0743-7/11/06.

Brief Announcement: Combine—an Improved Directory-Based Consistency Protocol

Hagit Attiya[*]
Technion, Israel
EPFL, Switzerland
hagit@cs.technion.ac.il

Vincent Gramoli
EPFL, Switzerland
University of Neuchâtel
vincent.gramoli@epfl.ch

Alessia Milani[†]
LIP6, Université Pierre et Marie Curie, France
alessia.milani@lip6.fr

Categories and Subject Descriptors

D.1.3 [**Programming Techniques**]: Concurrent Programming—*Distributed programming*

General Terms

Algorithms, Theory, Reliability

Keywords

Combining, overlay tree, stretch

1. MOTIVATION

Distributed applications in large-scale systems aim for good *scalability*, offering proportionally better performance as the number of processing nodes increases, by exploiting communication to access nearby data items. This paper presents a scalable *directory-based consistency protocol*: A node can introduce a new object by notifying the system (publish); to write an object, a node first acquires the object locally (move); to read an object, a node only has to get a read-only copy of the object (lookup).

Consistency protocols play a key role in *data-flow* distributed implementations of software transactional memory (DSTM) in large-scale distributed memory systems [2]. In these systems, remote accesses require expensive communication, and reducing the *cost of communication* with the objects and the number of remote operations is crucial for achieving good performance in transactional memory implementations. This stands in contrast with smaller-scale hardware shared-memory systems, providing faster access to local or remote addresses, where the critical factor seems to be the *single-processor* overhead induced by bookkeeping.

Several directory-based consistency protocols were presented in the context of DSTM implementation, e.g., [2, 5]. The general idea of a directory-based consistency protocol for a single object x, is to organize nodes in a tree and have nodes maintain a path towards a *sink* node. Initially, the sink is the node that creates the object. Then, the first request to modify x which reaches the sink will become the next to acquire the object after the current sink releases it, and the corresponding node will become the new sink. This is done by redirecting the followed pointers towards itself. To retrieve the value of x, a node has simply to follow the pointers and obtain a read-only copy from the reached node.

The message complexity of prior protocols deteriorates in the presence of concurrent requests. RELAY [5] uses a spanning tree, whose stretch can be quite high in common networks like rings and grids. Moreover, concurrent moves may cause nodes to communicate through the root of the tree, despite being close. Thus, its communication cost is proportional to the diameter of the spanning tree. BALLISTIC [2] uses an overlay tree augmented with shortcuts. Without concurrent requests, the cost is proportional to the stretch of the overlay tree, however, a mutual exclusion protocol is necessary to probe shortcuts in presence of concurrent requests, which increases its cost significantly. (See [1].)

We present a new directory-based consistency protocol, COMBINE, tolerating non-fifo message delivery and concurrent requests for the same object. COMBINE is designed to work in large-scale systems, where the cost of communication is not uniform, namely, some nodes are "closer" than others. Scalability in COMBINE is achieved by communicating on an *overlay tree* and ensuring that the cost of performing a lookup or a move is proportional to the cost of the shortest path between the requesting node and the serving node, in the overlay tree. Specifically, the cost of a lookup request by node p that is served by node q is proportional to the shortest path between p and q; the cost of a move request by node p is similar, with q being the previous node holding the object.

2. OVERVIEW OF COMBINE

We briefly describe COMBINE; more details, as well as proof of correctness, can be found in [1]. We assume that the cost of communication over single links forms a *metric*, that is, there is a symmetric positive *distance* between nodes, denoted $d(.,.)$, which satisfies the triangle inequality.

A key idea in COMBINE is that requests are combined as they pass through the same node. Originally used to reduce contention in multistage interconnection networks [4], *combining* means piggybacking information of distinct requests in the same message.

In more detail, a leaf node i can add a new object, look for its current value, or acquire it exclusively, as follows:

– A node i adds a new object x that it owns locally using a publish(x) request. This request sets a new pointer at each node located between i and the root of the overlay tree so that it points towards i.

[*]Supported by the *Israel Science Foundation* (grant number 953/06).
[†]Supported in part by ANR SHAMAN.

Brief Announcement: Paging for Multicore Processors

Alejandro López-Ortiz
David R. Cheriton School of Computer Science
University of Waterloo
Waterloo, Ontario, Canada
alopez-o@uwaterloo.ca

Alejandro Salinger
David R. Cheriton School of Computer Science
University of Waterloo
Waterloo, Ontario, Canada
ajsalinger@uwaterloo.ca

ABSTRACT

Paging for multicore processors extends the classical paging problem to a setting in which several processes simultaneously share the cache. Recently, Hassidim [6] studied cache eviction policies for multicores under the traditional competitive analysis metric, showing that LRU is not competitive against an offline policy that has the power of arbitrarily delaying request sequences to its advantage. In this paper we study caching under the more conservative model in which requests must be served as they arrive. We derive bounds on the competitive ratios of natural strategies to manage the cache, and we show that the offline problem is NP-complete, but that it admits an algorithm that runs in polynomial time in the length of the request sequences.

Categories and Subject Descriptors

F.1.2 [**Computation by Abstract Devices**]: Modes of Computation—*Parallelism and concurrency, Online computation*; F.2.m [**Analysis of Algorithms and Problem Complexity**]: Miscellaneous

General Terms

Algorithms, Theory

Keywords

multicore, chip multiprocessor, cache, paging, online algorithms

1. INTRODUCTION

In the last few years, multicore processors have become the dominant processor architecture. While cache eviction policies have been widely studied both in theory and practice for sequential processors, in the case in which various simultaneous processes share a common cache, the performance of even the most common eviction policies is not yet fully understood. In particular, there is almost no theoretical backing for the use of current eviction policies in multicore processors. Recently, a work by Hassidim [6] presented a theoretical study of paging strategies for shared caches in multicores or Chip Multiprocessors (CMPs). In a CMP system with p cores, a shared cache might receive up to p page requests simultaneously. Hassidim proposes a somewhat unconventional model in which the paging strategy can schedule the execution of threads. While in principle there is no reason why this cannot be so, historically the scheduler within the operating system concentrates in fairness and throughput considerations to determine which task should be executed while the paging algorithm focuses on which of the pages currently in cache should be evicted upon a fault.

In this work we assume a more conservative model, in which cache algorithms are not allowed to make any scheduling decisions but must serve all active requests. In this model, a paging strategy serves a set of p request sequences $\mathcal{R} = \{R_1, \ldots, R_p\}$ of total length n with a shared cache of size K. Requests can be served in parallel, thus various pages can be read from cache or fetched from memory simultaneously, and a fault delays the remaining requests of the sequence involved by τ units of time. We define as **FINAL-TOTAL-FAULTS (FTF)** the problem of minimizing the total number of faults, and as **PARTIAL-INDIVIDUAL-FAULTS (PIF)** the problem of deciding, given a request sequence $\mathcal{R}$ and bound vector $\vec{b} \in \mathbb{N}^p$, whether $\mathcal{R}$ can be served such that at time t the number of faults on each sequence R_i is at most b_i.

Without loss of generality we define a cache strategy as a combination of a possible partition policy, and an eviction policy, and compare the performance of natural strategies for FTF within this framework. We then study properties of the offline cache problem, both for FTF and PIF. We show that the latter is NP-complete, and give algorithms for both problems that run in polynomial time in the length of the sequences (and exponential in the number of sequences).

The performance of the cache in the presence of multiple threads has been extensively studied. From a theoretical perspective, researchers have studied schedulers and algorithms with good theoretical cache performance (See e.g. [3, 2, 4] and references therein). More directly concerned with cache replacement policies, *multiapplication caching* is studied in [1] in the competitive analysis framework. In this model sequences are not delayed upon faults, hence the problem is substantially different from ours. The work in [5] proposes an analytical model which predicts the performance of cache replacement policies in an application-by-application basis. Our work is concerned with arbitrary input sequences. Hassidim [6] considers minimizing completion time and shows that the competitive ratio of LRU is $\Omega(\tau)$. He also shows that computing the optimal offline schedule is NP-complete, and presents a PTAS for constant p and τ.