

# A Modular Measure of Competitiveness for Distributed Algorithms

James Aspnes\*

Orli Waarts†

The tool of competitive analysis has long been used to deal with nondeterminism in the form of unpredictable request sequences in on-line settings. The performance measure of an algorithm is its *competitive ratio*, the supremum over all possible request sequences of the ratio of the algorithm's cost to the cost of an optimal algorithm. This measure is often more practical than worst-case analysis, since it measures the ability of an algorithm to adapt to easy inputs as well as its ability to tolerate difficult ones.

In an asynchronous distributed setting, in addition to the unknown *input* (the sequence of future user requests), there is the unknown *schedule* (the timing of events in the system such as the arrival of messages in a message-passing model or the completion of low-level operations in a shared-memory model). Much of the work that has applied competitive analysis to distributed problems has the on-line and optimal algorithms compete only on the same input, generally hiding the details of the schedule in a worst-case assumption applied only to the on-line algorithm. More recently, the *competitive latency* model of Ajtai, Aspnes, Dwork, and Waarts [1], which measures how quickly an algorithm can finish tasks that start at specified times, has taken the approach of applying the same input and schedule to both the on-line and the optimal algorithms. This model has an advantage over its predecessors in situations where changes in the schedule have as big an impact on performance as changes in the input. In retrospect, this approach can also be viewed as an intermediate step leading to the approach adopted in this paper, in which the on-line and the optimal algorithms face the same schedule but may have different inputs.

We define a new measure of competitive performance for distributed algorithms, called *competitive throughput*, that measures the number of tasks that an algorithm can carry out with a fixed amount of work, without specifying the starting times of tasks. We maintain the split between user requests (the input) and system behavior (the schedule), but reverse the traditional approach by assuming a worst-case *input* and a competitive *schedule*. That is, we will assume

\*Yale University. Supported by NSF grants CCR-9410228 and CCR-9415410. E-Mail: aspnes@cs.yale.edu

†U. C. Berkeley. Supported in part by NSF postdoctoral fellowship DMS-9407652. E-Mail: waarts@cs.berkeley.edu

that both the on-line and the optimal algorithms must deal with the same pattern of failures and asynchrony, but that the user requests given to the on-line algorithm are chosen to minimize its performance while the requests given to the optimal algorithm are chosen to maximize its performance.

The above features of our model allow a modular construction of competitive algorithms. In effect, once we fix a schedule, we are doing worst-case analysis with respect to inputs, so we get the same modularity properties of standard worst-case analysis.

One difficulty remains. Traditional competitive analysis appears to forbid modularity: if  $A$  is an algorithm that uses a subroutine  $B$ , the fact that  $B$  is competitive says nothing at all about  $A$ 's competitiveness, since  $A$  must compete against algorithms that do not use  $B$ .

We overcome this problem by defining a notion of *relative competitiveness* such that if  $A$  is a  $k$ -relative-competitive algorithm that calls an  $l$ -competitive subroutine  $B$ , then the combined algorithm  $A \circ B$  is  $kl$ -competitive (the Composition Theorem). The essential idea (omitting some important technical details) is that for any choice of  $B$  the ratio  $\text{done}(A \circ B) / \text{done}(B)$  of tasks completed by  $A \circ B$  to tasks completed by  $B$  should be proportional to the same ratio for an optimal  $A^*$ . But since an optimal  $A^*$  might not call  $B$ , we instead consider two *independent* executions of optimal algorithms, one of  $A^*$  and one of an optimal  $B^*$  that implements the same object as  $B$ . If for every choice of  $B$ ,  $\text{done}(A \circ B) / \text{done}(B)$  is proportional to  $\text{done}(A^*) / \text{done}(B^*)$ , then we say that  $A$  is throughput-competitive relative to the object implemented by  $B$ . Intuitively, our Composition Theorem follows simply by multiplying out ratios; in practice, some care is needed to deal with short schedules and other pathological cases.

To illustrate the usefulness of our measure we take as a test case the collect primitive, in which each of a group of  $n$  processes must obtain the values of  $n$  registers. We first show that any collect algorithm with certain natural properties can be extended to a throughput-competitive implementation of a slightly stronger primitive, a *write-collect*. This result can be applied, for example, to the latency-competitive collect of [1] to obtain a throughput-competitive write-collect. We then use this write-collect primitive, together with our Composition Theorem, to derive competitive versions of many well-known shared-memory algorithms.

## References

- [1] M. Ajtai, J. Aspnes, C. Dwork, and O. Waarts. In 33rd FOCS, pp. 401–411, 1994.