1 Introduction.

Most applications of competitive analysis have involved on-line problems where a candidate on-line algorithm must compete on some input sequence against an optimal off-line algorithm that can in effect predict future inputs. Efforts to apply competitive analysis to fault-tolerant distributed algorithms require accounting for not only this input nondeterminism but also system nondeterminism that arises in distributed systems prone to asynchrony and failures. This chapter surveys recent efforts to adapt competitive analysis to distributed systems, and suggests how these adaptations might in turn be useful in analyzing a wider variety of systems. These include tools for building competitive algorithms by composition, and for obtaining more meaningful competitive ratios by limiting the knowledge of the off-line algorithm.

Like on-line algorithms, distributed algorithms must deal with limited information and unpredictable user and system behavior. Unlike on-line algorithms, in many distributed algorithms the primary source of difficulty is the possibility that components of the underlying system may fail or behave badly. In a distributed system, processes may crash, run at wildly varying speeds, or execute erroneous code; messages may be lost, garbled, or badly delayed. As a consequence, the worst-case performance of many algorithms can be very bad, and may have little correspondence to performance in more typical cases.

It is not surprising that the technique of competitive analysis [17] should be useful for taming the excesses of worst-case analysis of distributed algorithms. Of course, new opportunities and complications arise in trying to apply competitive analysis directly to fault-tolerant distributed systems. Distributed systems
have a natural split between the inputs coming from the users above and the environment provided by the system below—by exploiting this split it is possible, among other things, to build competitive algorithms by composition. On the other hand, because a distributed system consists of many individual components with limited information, one must be careful in how one defines the powers of the “off-line” algorithm so that the boundaries between these separate components do not become blurred. Work in this area has yielded several useful techniques for carefully controlling how much information the off-line algorithm is allowed to use.

Section 2 discusses distributed systems in general. Section 3 describes competitive analysis in its traditional form. Some variants on traditional competitive measures that are useful in models in which the nondeterminism naturally splits into two categories are described in Section 4. For these semicompetitive performance measures, composition of algorithms is possible, as described in Section 5. Examples of applications of these techniques to distributed problems are given in Sections 6 and 7. Finally, Section 8 discusses directions in which this area might profitably be extended.

A caveat: this chapter is concerned primarily with examples in the distributed algorithms literature of applying competitive analysis directly to nondeterminism in the underlying system. No attempt is made to cover the vast body of excellent work on on-line problems, such as distributed paging, load-balancing, routing, mobile user tracking, etc., that arise in networks and other distributed systems.

2 Distributed systems.

There are many forms of distributed systems, and a wide variety of theoretical models used to study them. However, there are two properties that show up in most distributed system models, which distinguish them from uniprocessor or parallel systems. The first and most important property is that a distributed system consists of more than one process, which may represent a real physical CPU, or might correspond to an abstract entity like a process or thread in a timesharing system. The second property is that the multiple processes of a distributed system are poorly coordinated—each runs its own program; the communication channels between them may be slow, expensive, and unreliable; and individual processes may crash, become faulty, or run at varying speeds. It is this poor coordination that distinguishes distributed systems from parallel systems, in which processes typically execute the same program in very close synchrony with each other using a powerful and reliable communication system. The line between parallel and distributed systems is not a sharp one, but a reasonable rule of thumb is that parallel systems are predictable; the same program with the same input running on the same parallel machine should give essentially the same results every time it is run. In contrast, distributed systems are riddled with nondeterminism—distributed algorithms are always at the mercy of an unreliable and sometimes hostile infrastructure, and must strive for robustness and consistency despite it.