10. Distribution

This chapter explores the issues affecting distributed systems and describes how Joule satisfies them. The solutions discussed assume the availability of the resource management tools described in the preceding chapter. By providing mechanisms to support process migration, as well as default policies, Joule supports the full spectrum of distribution regimes, from automatic distribution in which processes are automatically spread across multiple processors, to explicit distribution in which the programmer controls or influences the mapping from processes to processors, to untrusting distribution in which the programmer explicitly manages and adapts to trust boundaries and failure properties of the network.

10.1. Transparency

Transparency is the ability of a program written in Joule to function unaffected as it is stretched out across machines. This section will first describe the importance of transparency, then examine the primitives and the computational model to show how machine boundaries and communication lags can be invisible to an executing program.

10.1.1. Separation of distribution from correctness

Transparency allows programs to first be built to work, then be distributed without breaking the logic of the program. Because the assignment of processes to processors doesn't change the program, transparency also increases reliability and maintainability.

10.1.2. Adaptive distribution requires transparency

Adaptive distribution is the ability to write programs that migrate other, already-running programs to improve performance or adapt to the changing topology of a network. This adaptability requires that machine boundaries move in relation to the underlying program, without the program being changed. This transparency enables adaptive and automatic distribution, and also enables applying all the abstraction power of the language to the problem, including price-based competition among processors.

10.1.3. Channels stretch across wires

The semantics of Channels is such that they can be stretched across wires (with the inherent delays, etc.) without breaking.

10.1.4. Trust relationships are the same

The security system provides programs with ways of managing trust boundaries. Distributed systems simply introduce more trust boundaries, so the nature of the system stays the same, and programs will already be built to deal with the security problems revealed by distributed systems.

10.2. Failures in Distributed Systems

A continuously-operational open, distributed environment must remain robust in the face of many failure modes that are either not present or not obvious on single machines. This section explores many of them, and briefly describes Joule's solutions.

10.2.1. Node Failures

Node failures occur when a machine on the network fails. This section will describe the `Unavailable` exception which reports the failure and describes how to handle this exception. It will also describe using message plumbing to acquire control over the raising of this exception.

10.2.2. Network Partitions

A network partition is like a multiple-node failure except that the machines may return to service. Many applications can withstand the wait, so handling the return of access to a service is important. This section will describe the handling of the `Available` exception, which is reported when a service returns, and give examples.

10.2.3. Aberrant Behavior

Because of the Joule computational model, malicious and arbitrary behavior in a distributed system creates no new problems. Therefore, the security support deals with aberrant behavior of nodes in the distributed system. Further, the virtualizability of Joule channels allows them to be transparently encrypted between sites, so they can remain secure from eavesdroppers.

10.2.4. Node Amnesia

Since Joule is a persistent system, a particularly difficult form of failure is for a node to fail, and then revive in a previous state (from a checkpoint or backup). The issues here are complicated and subtle and will not be dealt with in detail in this document.

10.3. Explicit Distribution

This section will describe how a programmer can explicitly distribute a Joule computation. It will describe one particular distribution infrastructure, and how programs should interact with it.

10.3.1. Migration

This section will describe how to migrate processes between virtual processors in order to modify or improve the topology of a network.

10.4. Frameworks for Automatic Distribution

This section will build on the previous section to describe a framework in which programs can be automatically distributed (though not as well as a programmer might do), with no change to the program.

10.4.1. Simple Mechanisms

This section will describe a minimal strategy for automatically distributing processes to processors.

10.4.2. Stochastic/Heuristic

This section will describe stochastic and heuristic methods for load-balancing and distribution of processes to processors.

10.4.3. Agoric-Driven

This section will describe price-based strategies for load-balancing between processes using some of the agoric resource-management foundations.

10.5. Off-line Distribution

Occasionally-connected networks are those whose sites rarely talk to each other. This definition applies primarily to laptops and the networks to which they connect, and to networks that connect periodically to transmit updates (for example, USENET links). This section will describe how Joule distributes successfully over occasionally-connected networks.