COMPSCI 630 Systems **Spring 2017**

Lecture 10: Little's Law, Flux, Trusting Trust and The Confinement Problem

Lecturer: Emery Berger *Scribe(s): Vikram Pawar*

10.1 Performance Analysis

Different machines have different performance bounds like the CPU, I/O and memory. For example, memory-intensive applications like in-memory databases could have memory bus saturation. So how does one determine what the performance bottlenecks for any system are? One way is to generate synthetic load, but this is difficult. The empirical approach is to generate a trace of the program and run it through a discrete event simulator. Several different kinds of simulators exist; eg: chip manufacturers use chip simulators that they run with program trees.

Another method is to use queuing theory to simulate the system as a system of queues. There is a whole branch called operations research based on queuing theory.

10.1.1 Queuing Theory

We model everything in the world as a queue and a server and combine them into queuing networks.

If the arrival rate is more than the service rate, the queue will grow without bound. If people are waiting to served, the system is said to be 'backlogged'. In a stable system, the arrival rate equals the departure rate. If the service rate is higher than the arrival rate, then the system is starved. A simple example of a queuing system is a threadpool. Each thread can serve only one request at a time, and thus there is a bound on the number of requests that can be handled at any given time.

10.1.2 Little's Law

$L = \Lambda\ W$

Little's law states that in a queuing system, long-term averages of the Length of the queue(L), the waiting time(W) and the arrival rate(A) are related according to the above relationshp. This holds for any arbitrary distributions over L, Λ, and W. This law holds for complicated systems, which is why it is so useful. The standard approach in queuing theory is to model the arrivals and service rate as Poisson processes (called Markovian). In Kendall's notation, this is M/M/1 where the last '1' is the number of servers.

10.1.3 Flux

This presents us with the problem of figuring out if the model of the system is accurate. This is called the model validation problem. Flux addresses this problem with a Discrete Event Simulator and a Queuing Network Model generated from the program. Flux also supports thread-based and event-based runtimes. The thread-based runtime may spawn a new thread for every request or use a thread-pool. The event-driven runtime is complicated to implement due to blocking function calls. The event-driven runtime passes any

blocking calls to a special handler that executes them asynchronously. When the blocking call returns, the event-based runtime receives a signal and the event is queued again.

An example of an event-based runtime is Javascript, which uses the continuation-passing style of registering callbacks for blocking function calls.

10.2 Security

10.2.1 Reflections on Trusting Trust

A backdoor is a method of getting unauthorized access to a system by embedding a secret program or modifying part of a program undetected. A famous example of this is Ken Thompson's modification of the C compiler, which inserted a piece of code into the unix login program that allowed Ken to login to any unix system. In the paper, Thompson points out that the only way to trust code is to build a trusted computing base. This base has to be small because that is the only way that the code can be manually inspected in a reasonable amount of time.

Another approach could be formal verification. However, this requires that the verification system be trusted, which itself a complicated system and very difficult to verify.

10.2.2 Confinement Problem

The confinement problem is the problem of making sure that a program is unable to leak confidential data to a third party. A program can leak data using a number of methods. Some of these methods are fairly direct like communicating directly to another program, writing to a permament or temporary file and letting another process read it, encoding information in a bill or any data like the time used for rendering the service. In general, these direct methods can be easy to deal with. Covert channels are more difficult to subvert since these involve indirect means of transmitting data. For example, a program can transmit boolean values by a number of means like changing the system temperature in a systematic way by taking different branchesof execution. Legitimate and covert channels can be blocked by masking, which means that a program must be confined to allow its caller to determine all its inputs into legitimate and covert channels. A supervisor program would have to ensure that the confined program adheres to specifications by, for example, ensuring that all branches take comparable amounts of time. Since it is expensive to completely block all channels, a tradeoff is usually the best alternative where the capacity of covert channels is bounded.‘