

Debugging A Large Node Cluster

Intro

Debugging a Large Node Cluster

So far, node is delivering pretty well on it's initial promise:

- Very good performance on a single CPU core
- Inexpensive to hold open lots of network connections
- You get to write network programs in JavaScript

There is always this nagging question:

- But what about my other CPUs? Isn't all modern software multi-threaded?
- Probably shouldn't have said that node was more "scalable", since it is at best misleading, and at worse totally wrong.

Most of us True Believers have always said:

- Event loops are good, they maximize efficiency
- Just get more nodes
- network connections are cheap
- each instance is fairly cheap
- eventually you'll run out of cores and need to move to multiple machines anyway
- BTW, node has Unix sockets and sendFd, so go make things like Worker and Cluster

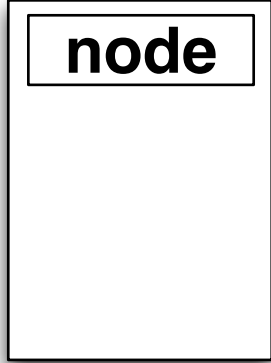
Debugging a single node process is hard enough

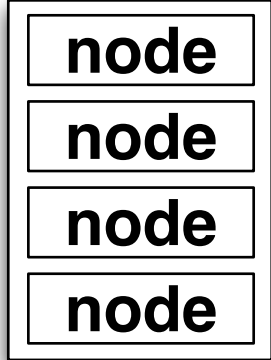
- You can use node inspector, which is amazing, but still lacks a lot of things you want
- Since most useful node processes handle lots of connections, pausing the process to debug it often breaks the application
- Even if it doesn't, you still can't "step" through async code the way you'd like.
- You can also use node's builtin interface to the V8 debug API if you like.
- These are fine if you can stop the process, but that's a lot harder to do on a running server.
- Or, you can do what I do, which is log a lot.

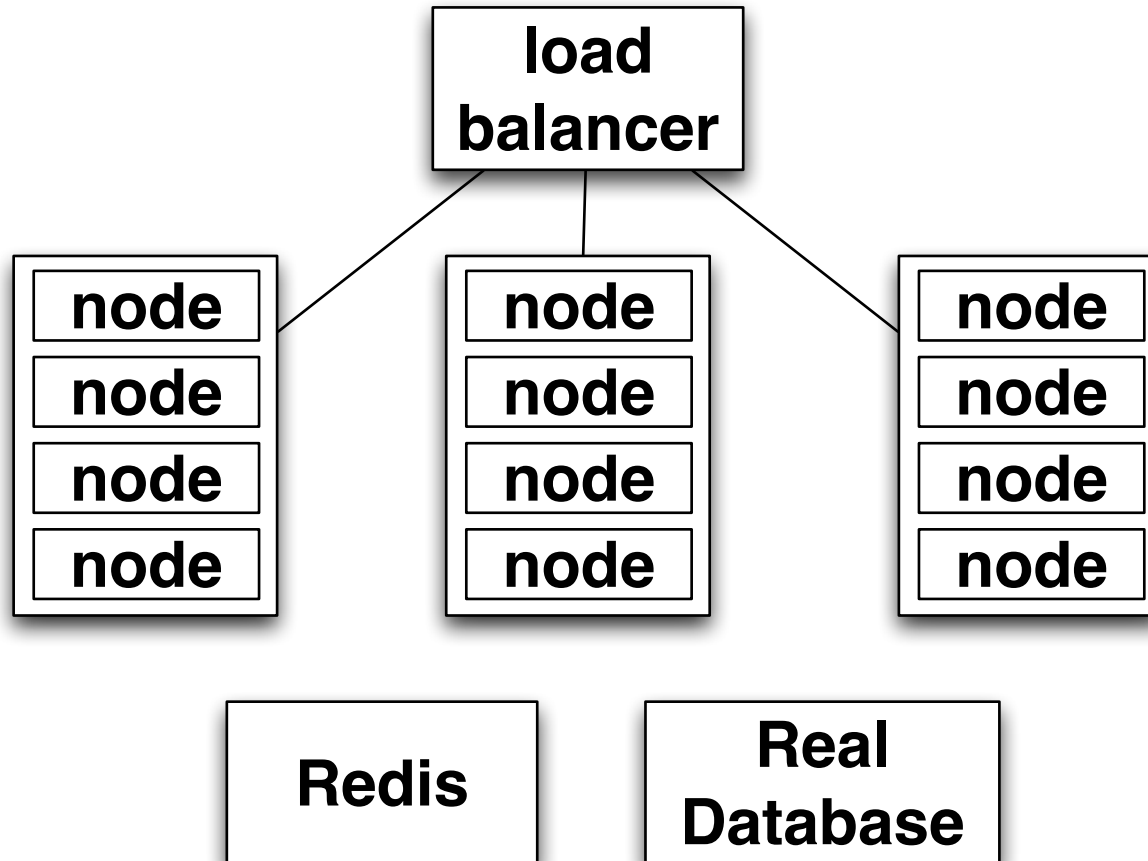
But don't forget, very good performance on a single CPU core, lots of connections, JavaScript, cheap.

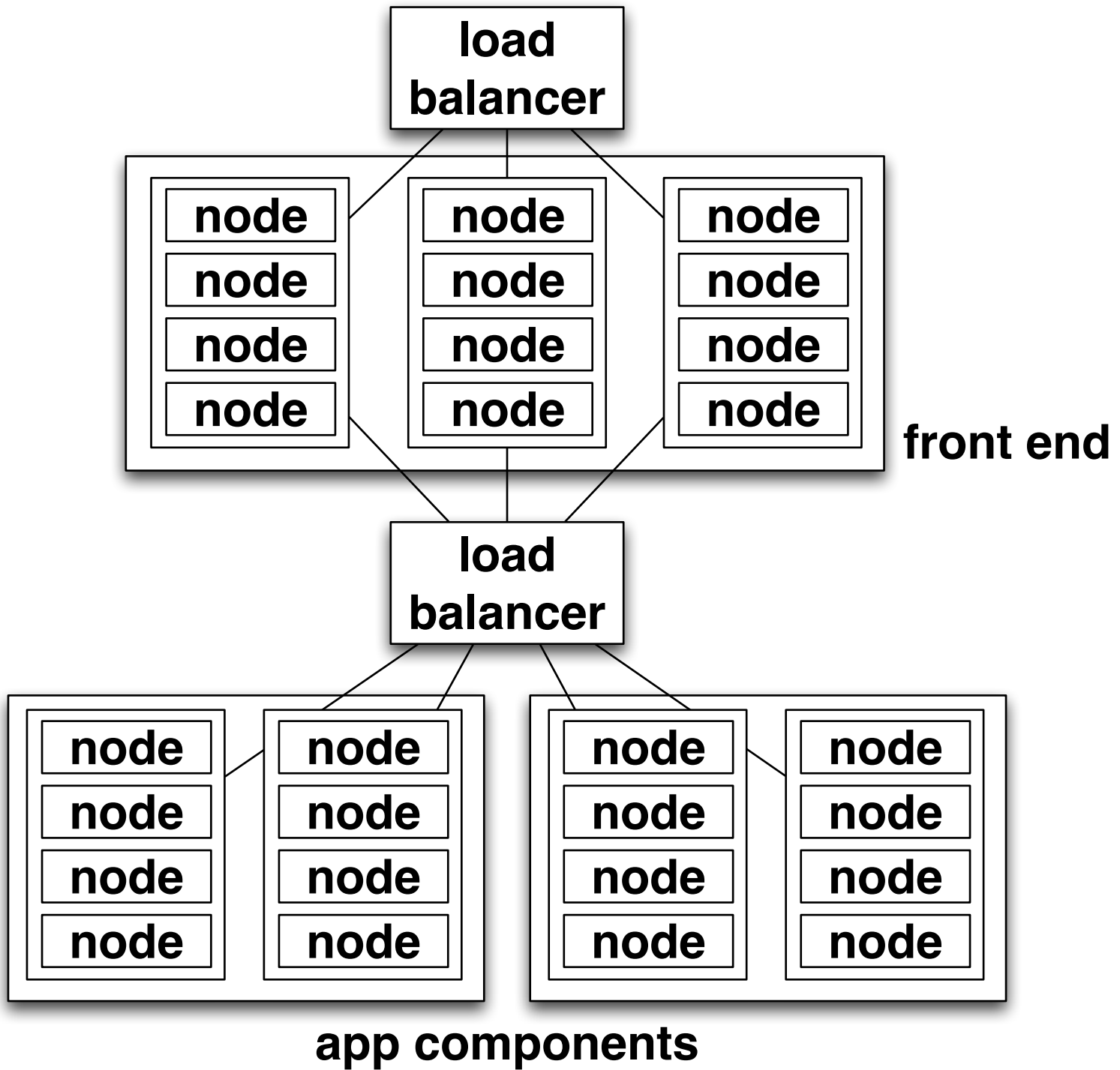
You can actually debug a single node process. It may be ugly, but it's a pretty approachable problem.

Show progression of diagrams.









OK, so let's start expanding to multiple computers. There are lots of ways to do this, depending on your application.
For most servers, you'll end up with a pool of processes behind some kind of load balancer
Maybe some shared state in Redis, or possibly memcached if you are someone other than me.
Many people also use other database technologies, and there are lots of wonderful choices these days.

Things are pretty much the same now, right?

You might expect that.

The system is built out of these simple building blocks

You are very responsible, you follow all of the rules, and you have good tests, on a testing cluster even.

As soon as you add multiple machines into the mix, things can get strange.

loopback networking behaves somewhat differently than Ethernet

in a shared hosting scenario, each instance may have very different performance, depending on other tenants.

try as you might to practice good ops, sometimes configs are different

you probably aren't going to build a test lab that's EXACTLY the same as your production environment

The biggest difference of all though, is the shape of real user traffic.

It is impossible to accurately simulate

Even if you can generate the same amount of traffic for testing, you can't simulate the shape

You also cannot simulate the multitude of network elements involved between your users and your servers.

You will find really strange bugs that open happen in production, and only happen with live user traffic.

This is a really big problem.

This is not something that node itself can fix, because it's a fundamental problem of horizontal scaling.

Some other environments like Erlang sound like they make this problem easier

I believe they do address some concurrency and coordination issues, but that's not what this is.

This is a problem of distributed debugging

What we need is the same set of tools for just barely being able to debug a single node, except we need them to work across a large cluster of nodes.

I do not have this. However, there are some things you can do to get closer to this goal.

If you have worked on a very large scale operation before, you probably already know this better than I do.

What makes this relevant for node is how quickly these issues come up.

I have not done all of these things yet, but these ideas, like Xena, Warrior Princess, have been forged in the heat of battle.

Aggregate your logs somewhere, somehow.

- Need to be able to see a timeline of events happening on multiple machines.

- Must be searchable, at least by grep

- Realtime access is really useful

- Lots of options for this these days

Collect stats on everything, even if you aren't sure exactly how to use them, or even if you don't ever look at them.

- Stick them in Redis, Cacti, whatever.

- Really check out "metrics"

- Collect and store all of this stuff, and hopefully graph it somehow.

- Even if you haven't yet managed to visualize all of your stats, at least capture them.

Pushing code changes things

- Some bugs may be caused by a sequence of events putting the program into a certain state

- When you restart to add logging, you lose that state, and may make the bug harder to find.

This is where JavaScript comes in. You actually can change your program without restarting it using the REPL.

- You can poke around and inspect things, as long as you expose them to the REPL scope.

- Even this is still running new code inside your program.

- You can explicitly invoke functions that you've left in your code.

- You can even add new functions to your program that'll be run in different places.

Show REPL demo