X-Trace

Spring 2017

Contents

1 X-Trace

X-Trace is a tracing framework for distributed systems. It works like a logger does, but organizes your log statements visually in *causal order* instead of printing them in temporal order on the terminal. These traces can make it much easier to debug and view your systems than just reading through log statements. X-Trace is described in more detail in Chapter 3 of Rodrigo's Ph.D thesis [1], which you should feel free to read!

The basic principle of X-Trace is that log statements are assigned "event IDs", unique identifiers for each log statement. Whenever you log an event, you send the log message itself, its event ID, and the event IDs that preceded or caused it. The central X-Trace server organizes these statements into a graph, and provides a partial ordering of the events. Events are organized into "Tasks", or traces, that represent different independent processes in your programs. Each task can have lots of events in it, as well as several tags detailing what it represents. Deciding when to create new traces, and when to log events, is a big part of using X-Trace effectively.

2 Installing X-Trace for Go

First, you will need to download the X-Trace server backend here: `http://cs.brown.edu/courses/cs138/s17/demos/xtrace-backend.zip`. (If you're working on a department machine, it's available as `cs138_xtrace_backend`). All the X-Trace functions will fail silently if there isn't a server running.

[1] `https://www2.eecs.berkeley.edu/Pubs/TechRpts/2008/EECS-2008-167.pdf`

After you download the server, you can run it with either the `backend` or `backend.bat` executable in the `bin/` directory. It will create an `xtrace-data` in your working directory that will store all the events and traces as JSON data.

To use X-Trace in Go, we need a way of keeping track of event IDs between log statements that is easy for the programmer. The best way to do this would be to give each goroutine some variables that only it could see or modify. Go doesn't support this by default, so we're providing a program that makes a small modification to the Go runtime to support this. To install this, run:

```
$ go get github.com/brown-csci1380/tracing-framework-go/cmd/modify-runtime
```

```
$ modify-runtime
```

You should then see a large printout detailing what was changed, ending with `Creating local access methods.....Done`. If you're working on a department machine, we've provided an already modified runtime package which you can access by changing your `GOROOT` environment variable; run `export GOROOT=/course/cs1380/pub/138goroot`. After that, you're ready to start using X-Trace!

3 X-Trace Functions and API Detail

3.1 Demos

This section will reference several demo Go files, which you can download and run yourself at `http://cs.brown.edu/courses/cs138/s17/demos/xtrace.zip`.

3.2 Basic X-Trace Functions

The X-Trace package you will need to use is `github.com/brown-csci1380/tracing-framework-go/xtrace/client`, which we usually import as "xtr".

1. `func Connect(server string) error`
   Connect must be called before any other X-Trace functions. It establishes a connection to a running backend server, and returns an error if it can't be established. You should call this during one of the `init()` methods of your package.

2. `func Disconnect()`
   Tears down the connection to the backend server. Should only be called once (usually at the end of `main()`).

3. `func NewTask(name string)`
   Creates a new task and tags it under the given name. All log statements before the next call to NewTask will be grouped under this task. Needs to be called once before you log anything.

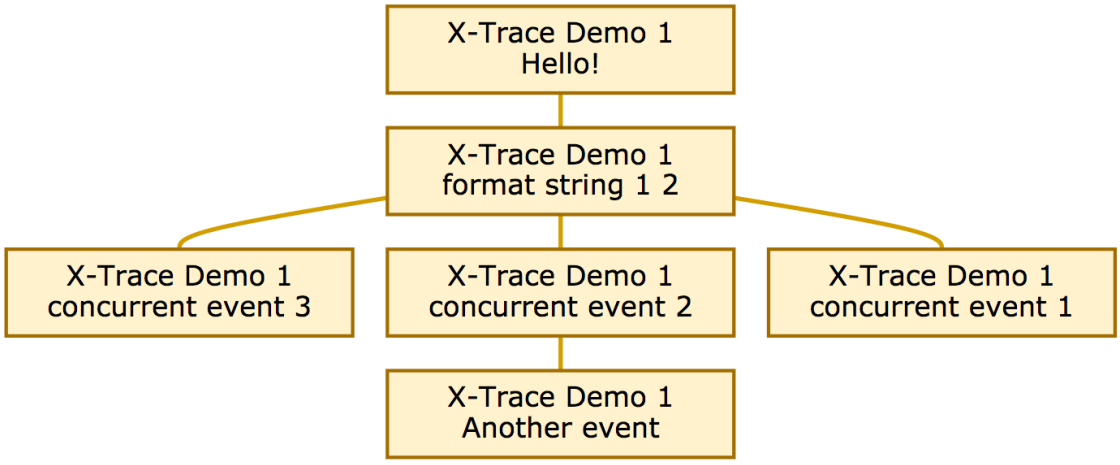
4. `func AddTags(tags ...string)`
   Adds the given tag(s) to the current task, which you could use to mark a task as having encountered an error, completed a step, or anything else you might want to differentiate it from similiar tasks. Doesn't take effect until the next Log statement.

5. `func Log(message string)`
   Logs the given message to the X-Trace server, noting which event(s) preceded it.

6. `func Logf(format string, args ...interface{})`
   Works like Log, but accepts arguments like Printf.

7. `func XGo(f func())`
   Runs the given function in a new goroutine, but causes it to inherit the events/task from the current goroutine.

You can see these functions in use in the `xtrace-demo1.go` file. Run this file with `go run xtrace-demo1.go` while the X-Trace server backend is running, and point your web browser at `http://localhost:4080` to see the results. The "Animated" trace for the code you just ran should look like this:



Also node that, instead of using `XGo`, you can use the normal `go` statement, and then rewrite your code using the `xtrace-rewrite` tool, available by `go get github.com/brown-csci1380/tracing-framework-go/cmd/xtrace-rewrite`. You can run it after installation with `xtrace-rewrite <file or directory>`, which will replace all `go` statements in the given files with equivalent `XGo` statements.

3.3 Using with Built-in Loggers

You can also use X-Trace with the existing logging framework in Go. The `MakeWriter` function wraps an existing object that implements `io.Writer` (i.e., `os.Stdout` or `os.Stderr`), and returns a new `io.Writer` that writes to both the underlying writer and logs to the X-Trace server. See the `xtrace-demo2.go` file for a similar example as the first, but using a logger called `Trace` to log events instead of the `xtr.Log` function directly. Run it with `go run xtrace-demo2.go`.

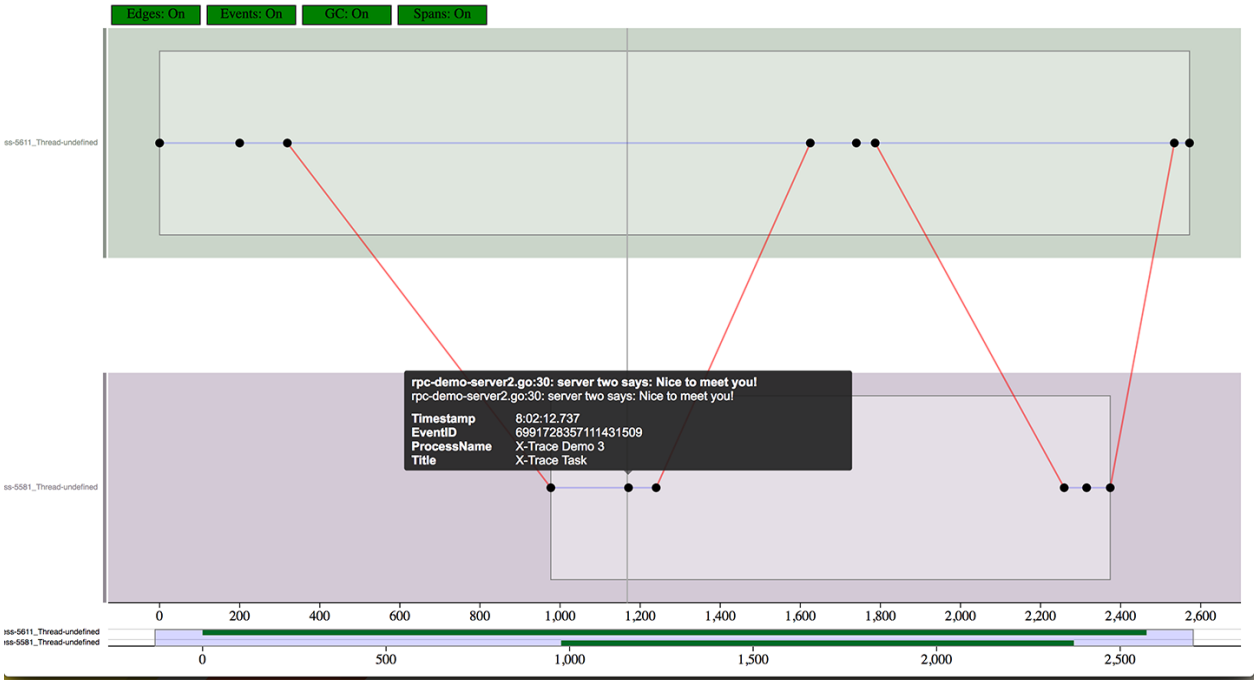
1. `func MakeWriter(writer ...io.Writer) io.Writer`
   `MakeWriter` can be used as an argument to `log.NewLogger()` to wrap one or more existing writers (like the `io.MultiWriter` or unix `tee` command) into one writer that prints to all of them, in addition to X-Trace.

4 RPCs

One challenge for using X-Trace in a distributed system is ensuring that event IDs can make it across network communications, so that events that follow on a different server are still part of the same task and display correctly. We've also provided a few functions to accomplish this for you, in the `"github.com/brown-csci1380/tracing-framework-go/xtrace/grpcutil"` package.

1. `var XTraceServerInterceptor grpc.UnaryServerInterceptor`
   XTraceServerInterceptor is a function that propagates X-Trace metadata through gRPC calls on the server. Pass it as an argument to `grpc.NewServer` with `grpc.UnaryInterceptor`, e.g. `srv := grpc.NewServer(grpc.UnaryInterceptor(grpcutil.XTraceServerInterceptor))`.
2. `var XTraceClientInterceptor grpc.UnaryClientInterceptor`
   XTraceClientInterceptor is a function that propagates X-Trace metadata through gRPC calls on the client. Pass it as an argument to `grpc.Dial` with `grpc.WithUnaryInterceptor`, e.g. `cc, err := grpc.Dial(addr, grpc.WithUnaryInterceptor(grpcutil.XTraceClientInterceptor))`

As long as you pass those options to `grpc.Dial` and `grpc.NewServer`, X-Trace events and task IDs will travel with your RPC calls. You can see a demo of this by first running the `rpc-demo-server.go` with `go run rpc-demo-server.go`, and then in another terminal, running `go run xtrace-demo3.go`. Look for the "demo 3" trace, and view the "swimlane" diagram instead of the other two views. You should see the log statements organized in a familiar way:



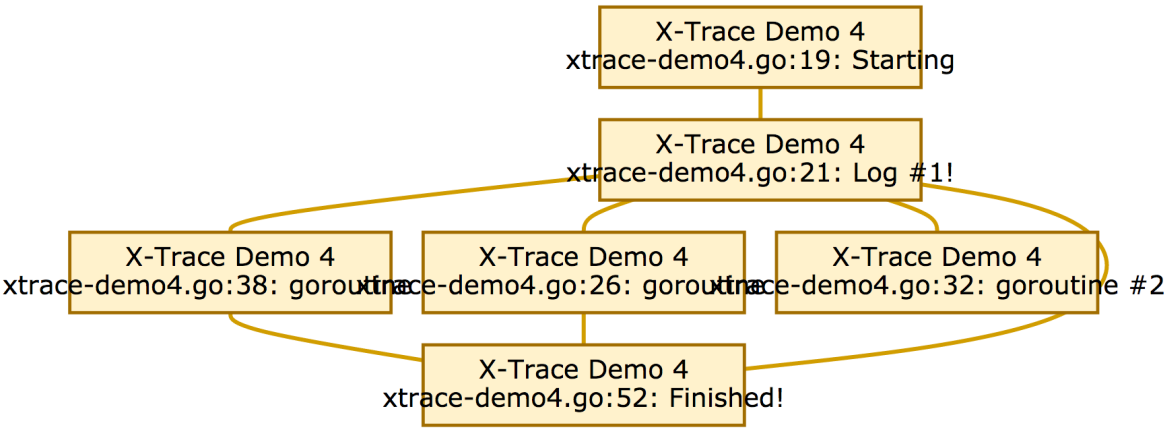
4.1 A Note About Channels

There's an additional challenge in making sure X-Trace events make it across channel communications. This isn't necessary to get good traces in a lot of our uses, but if you want to use it, we have also provided two methods to use.

1. `func SendChannelEvent(channel interface{})`
   Should be called immediately before sending on a channel, passing the channel you're about to send on as the argument. Informs whoever recieves from the channel which goroutine (and event, task) sent on it.

2. `func ReadChannelEvent(channel interface{})`
   Should be called immediately after receiving from a channel, passing the channel you just read from as the argument. Adds an event from the goroutine which sent over the channel to the current goroutine.

You can run a demo of this with `go run xtrace-demo4.go`, which shows how collecting the events from the channels produces a different look on the last log statement.



5 Tapestry

To use X-Trace in tapestry, after installing the "tracing-framework-go" package and modifying your runtime, you need to uncomment all the "uncomment for xtrace" blocks of code that we've left in for you. Then, feel free to use the "Trace" logger to log events to X-Trace, and use the "xtrace-rewrite" tool to swap out "go" statements for "xtr.XGo" statements where you feel it's appropriate.

Please let us know if you find any mistakes, inconsistencies, or confusing language in this or any other CS138 document by filling out the anonymous feedback form:

`http://cs.brown.edu/courses/cs138/s17/feedback.html`.