DisTcl - Distributed Programming Infrastructure for Tcl

Colin Macleod

colin_g_macleod@yahoo.com

CGM on Tcl wiki and chat

Introduction – Distributed Processing

There are many computing tasks which require multiple machines or at least multiple processes to execute.

Unless these tasks are entirely independent we then need some means to communicate data and coordinate processing between multiple machines and processes.

There are of course many frameworks in existence to manage this, but they are often complex and difficult to use.

Distributed Processing - issues

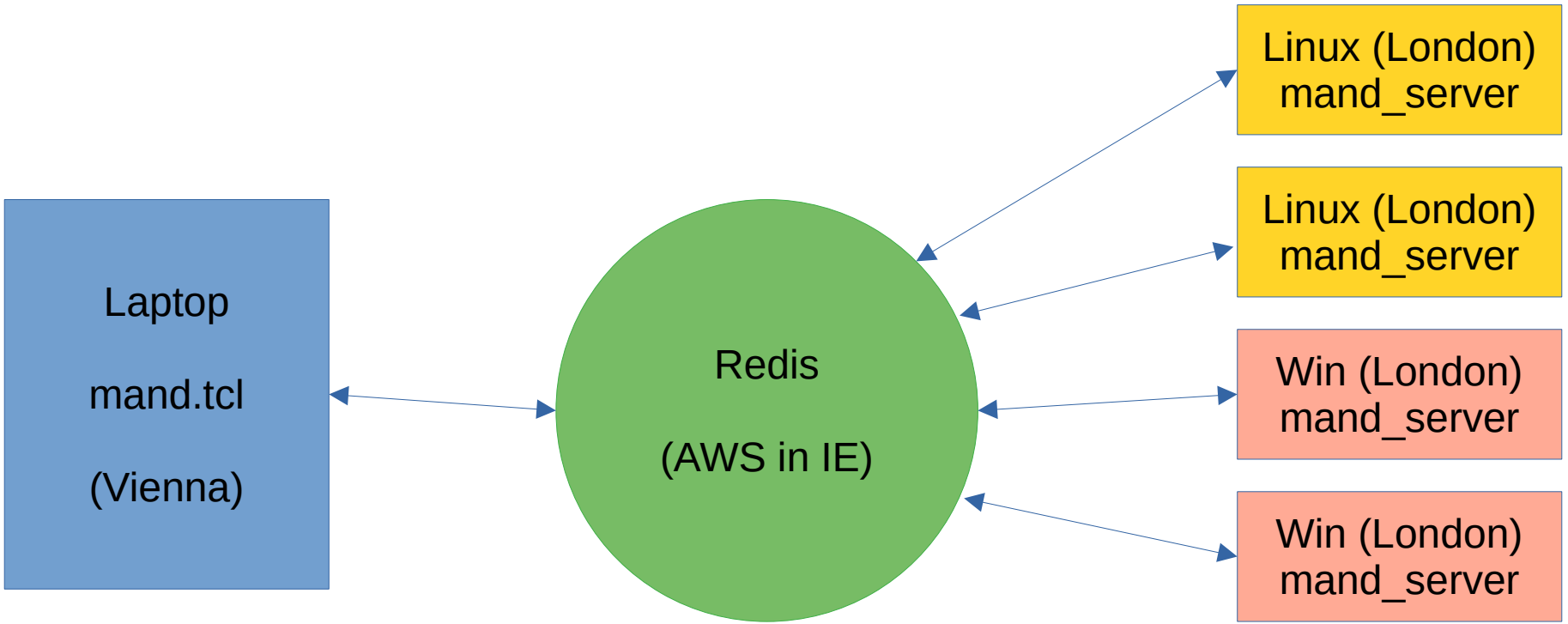
- Each separate process and machine needs to be able to find the others which it needs to read data from or send data to.
- We want the processing load to be spread evenly across the available machines and processes.
- It is highly desirable to be able to bring up or shut down additional processes/machines to respond to changing demand or maintenance needs .
- When the same results may be needed more than once we would like to cache them for reuse by any other process or machine.

Distributed Processing – issues 2

Existing frameworks I have worked with require complex and error-prone configuration to achieve these objectives.

DisTcl aims to achieve the same ends in a simpler and more flexible way. It does this by using the Redis database system for inter-process communication and caching.

Demonstration – Mandelbrot Set



Mandelbrot Server

```
proc slice {Cheight Rmin Rscale Imax Iscale start stop col_ver} {
…..
proc do_mand {what args} {
    switch $what {
        slice {return -secs2keep 300 [slice {*}$args]}
        square {return -secs2keep 600 [expr {[lindex $args 0] ** 2}]}
        hello {return "Hello from [pid] on [info hostname] running $::tcl_platform(os)"}
        default {error "Don't know how to '$what'"}
    }
}
distcl::serve redis mand do_mand
```

Minimal Test Client

mand_client.tcl :

```
#!/usr/local/bin/tcl
source distcl.tcl
source redis-cloud.tcl
puts [distcl::get redis mand {*}$argv]
```

Redis - introduction

Redis is a well-known in-memory database system often used for caching. Its name means REmote DIctionary Server. A Redis server provides various forms of data structure which can be accessed by network connections from other processes and machines.

Redis was created by Salvatore Sanfilippo (a.k.a. Antirez) who started by writing a prototype in Tcl to check the feasibility of his design and then reimplemented it in C. One can still see Tcl influences in Redis, e.g. there is a command LINSERT which operates on lists in a very similar way to [linsert] in Tcl.

Redis - licensing

Recently there has been some controversy because the company which now controls Redis changed the license of the code to one which is not generally regarded as Open Source.

In response to this several forks and compatible alternatives have appeared. DisTcl should be able to work with any of these, though I have not yet tested this.

Redis - structure

Redis can be considered as a "noSql" database. There is no schema to impose a structure on the data. Instead one simply creates and uses data items as required on-the-fly.

There are forms of data which correspond to simple variables, lists and arrays in Tcl and some more advanced forms also. To interface Redis with Tcl I am using a package called Retcl written by Pietro Cerutti.

DisTcl - Design

- DisTcl organises computing facilities into named services.
- A service responds to some set of messages to perform certain tasks and return corresponding data.
- A service can be implemented by one or more server processes, and used by one or more client processes.
- A process which is a server for one service could also be a client for another service.

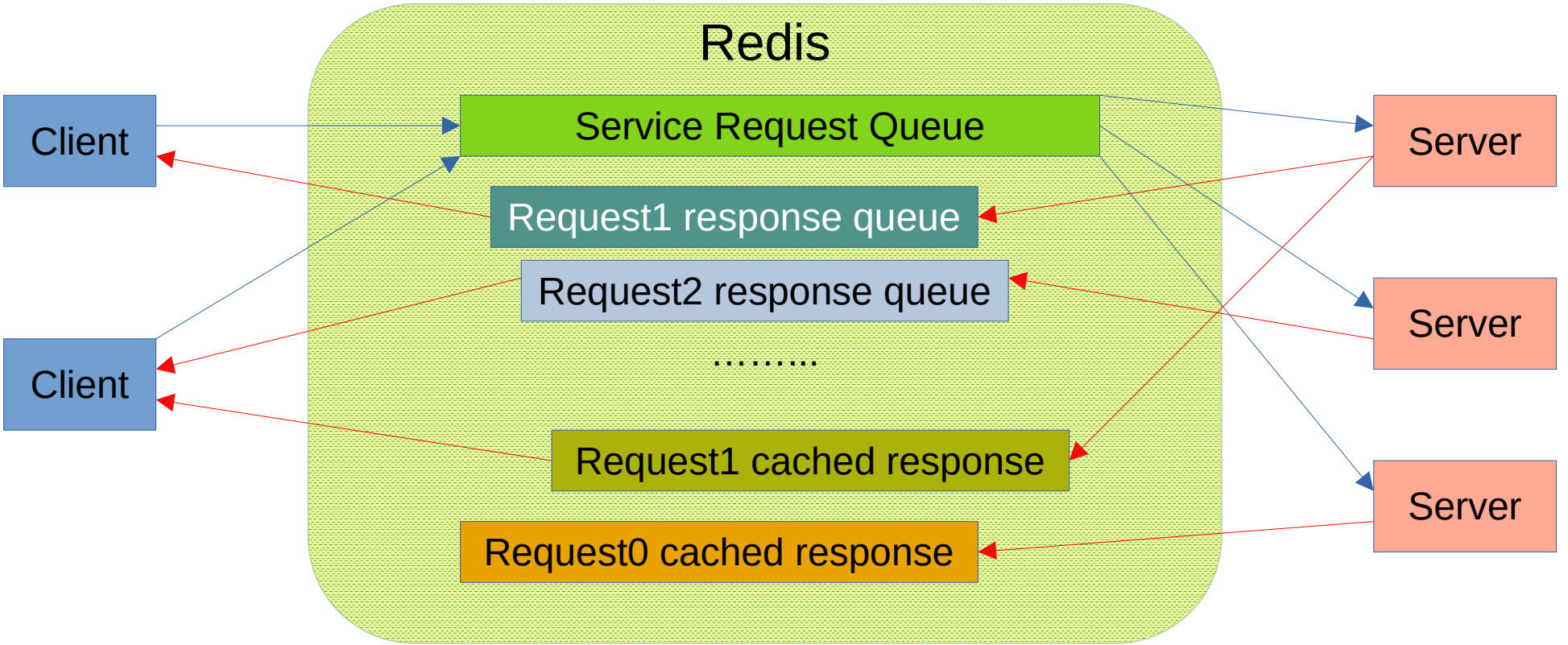
DisTcl – Design 2

- DisTcl clients and servers connect to a Redis instance which then relays requests and responses between them, and also caches results which are stable enough for this to be appropriate.
- This communication uses Redis lists as pipelines of data, with different clients and servers writing to a pipe and reading from it. Redis also provides more sophisticated forms of queue, but simple list-based queues suit DisTcl's needs.

DisTcl – Design 3

- A client for service ABC will write requests to a corresponding Redis queue.
- A server for ABC will pull requests from that queue, process them and write results back to another queue which the client will read from.
- If the result is one which can be cached, the server will also store it in Redis so that future requests for this item can be served directly from the cache.

DisTcl – Redis structures



DisTcl – Design 4

- So participating processes only need to know how to connect to the Redis instance, they don't need to know the locations of other processes.
- The Redis list operations ensure that each request will only be read by one server, the next which becomes available.
- Additional servers can be started or stopped as required and we get a simple form of load-balancing across them, with no other configuration needed.

DisTcl – Design 5

Usually when a client makes a request it then waits for the response before proceeding. However sometimes the client knows that it will need multiple requests, which could be processed in parallel if there are free servers available.

To take advantage of this parallelism the client can make "prefetch" requests for all these items, without waiting for the results. If there are servers free, they will start processing these requests and storing the results in the Redis cache. Then when the client actually requests each item, some or all of them will already be in the cache and can be returned immediately.

DisTcl – use in Newsgrouper

- Newsgrouper is a web site I'm working on which provides access to Usenet newsgroups without the user needing an NNTP client. I use DisTcl as part of the infrastructure behind this.
- There is a single service called "ng" which provides commands to get the list of articles for a group, to get the text of a specific article etc.. The server for this is a script which uses the nntp package from Tcllib to pull data from a Usenet server. Since the server permits up to four concurrent connections I can run 4 instances of this script in parallel.
- The client for the "ng" service is a tclhttpd web server, which can serve requests concurrently in multiple threads, so each thread acts as a separate client.

DisTcl – General Points

- Other languages - There is nothing Tcl-specific in how DisTcl works. Clients and servers using the same protocol could easily be implemented in other languages, though they might need a little more code to serialise/deserialise data, which Tcl does automatically.
- Redis tools - Using Redis means that various existing tools can be applied. E.g. redis-cli lets you run commands interactively to examine or modify data, and its MONITOR command lets you see all updates as they happen.
- Other data - Of course once you have Redis, it may be convenient to use it to store other data. Newsgrouper also uses Redis to store user-specific data like which articles a user has read, outside of the DisTcl system.

DisTcl - Scalability

- For busy applications the Redis server could become a bottleneck, limiting throughput.
- Redis supports Clustering of servers to allow an instance to scale beyond the limits of a single machine. I have not yet examined in detail whether DisTcl could work on a Redis cluster.
- However if multiple services are being used, an easy way to scale up would be to allocate different services to separate Redis instances.

DisTcl – Future Directions

- There are many ways in which the existing DisTcl code could be improved and optimised.
- Redis supports running scripts in the Lua language inside the server. So one improvement would be to convert sequences of Redis operations into Lua scripts. This would reduce network traffic and also ensure that such sequences of operations are executed atomically.
- Last year I presented a simple system for reactive computing in Tcl, called ReacTcl. This provides on-demand computing and memoisation of results within a single process. DisTcl has some similarities but operates between processes and machines. It might be useful to combine ReacTcl and DisTcl into a unified system, but I have not done any serious work on this yet.

DisTcl – Questions?