University of St Andrews | FOUNDED 1413

ORACLE for Research

Moving a scientific computing system to the cloud

Simon Dobson and Peter Mann
Complex and Adaptive Systems Group
School of Computer Science, University of St Andrews UK

simon.dobson@st-andrews.ac.uk
https://simondobson.org
http://mastodon.scot/@simoninireland

pm78@st-andrews.ac.uk
https://peterstandrews.github.io/
https://twitter.com/PMannStAndrews

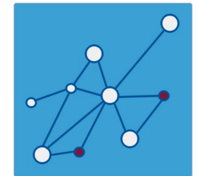
Introduction

- Complex and adaptive systems
  - Microscale interactions give rise to difficult-to-predict macroscale behaviours
  - Disease epidemics, protein interactions, metal fatigue, ...
  - Lots of numerical simulation of stochastic processes

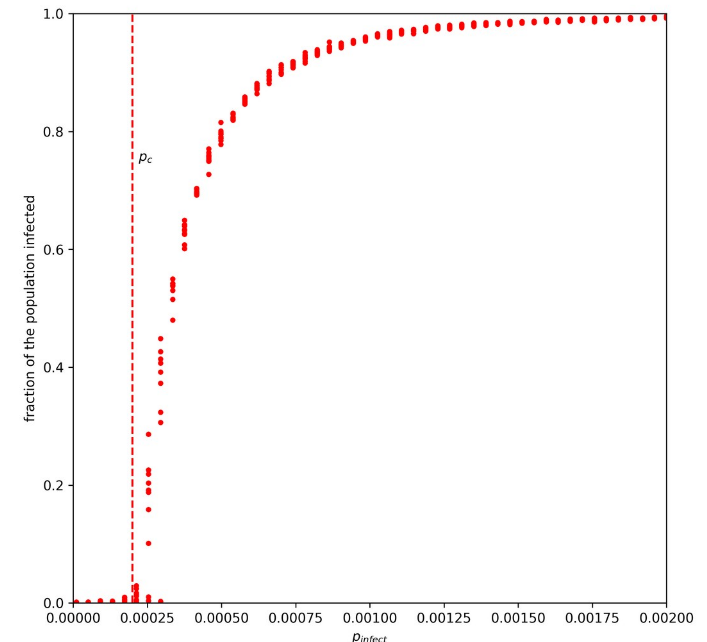
- Challenges and opportunities in moving to the cloud
  - Architectural flexibility
  - Autoscaling interactive code
  - Performance tuning

epydemic

- Process simulation on networks and other structures
  - Epidemics, synchronisation, percolation, …
  - Networks and (soon) simplicial complexes
  - Maths helpers, network generation, ...
  - Gillespie simulation



- Simulating stochastic processes
  - Large networks, lots of repetitions (all independent)
  - Each individual experiment takes from <5s to >20mins



Why move to the cloud?

- Scalability
  - Tackle larger, higher-resolution simulations
  - (Possibly) work with significantly larger networks (using graph databases' query optimisation)
- Modern virtual infrastructure
  - Run on someone else's computers (*a.k.a.*, the cloud)
- Easier sharing and deployment
  - Spin-up instances of the entire infrastructure

We're grateful for support from Oracle for Research
in the form of cloud credits and expertise to call on

Issue 1: Maintaining flexibility

- Originally tried to replicate our "classical" cluster
  - `ipyparallel` for comms, `redis` for synchronisation
  - Not the right model for modern platforms
- Embrace virtual infrastructure

  - Containerised implementation of our scientific code (unchanged apart from a small web API)
  - Gateway API as a single point of contact
  - RabbitMQ co-ordinates all the work requests and results
  - Works *with* (rather than fighting against) the model
  - Deploy locally for debugging, same interfaces

No dependencies, so easy to re-purpose within other structures

Architecture

Can add extra steps in the message processing inside a synchronous API

Messages are consumed to generate synchronous experiments, whose results are returned as messages

Synchronous API call turns into a message that's queued for delivery to some instance of the compute engine

Can scale the number of compute engines and the broker architecture integrates them transparently

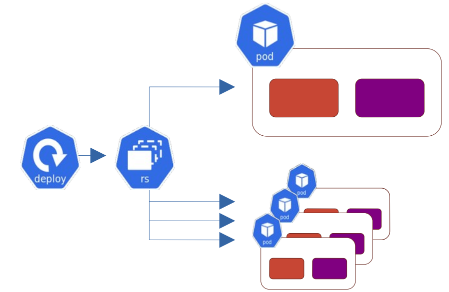
Another API call collects all results that have been resolved

Web API

Shim to map messages to and from API calls

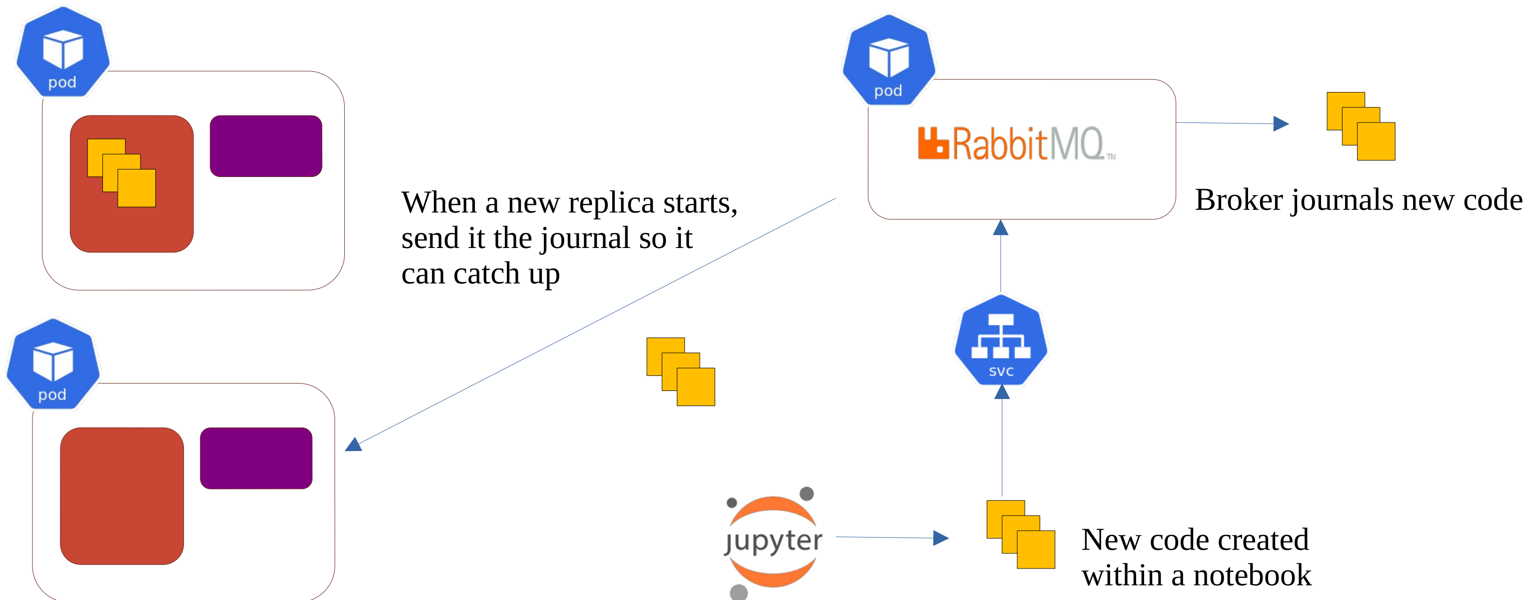
Issue 2: Interaction and autoscaling

- (Horizontal) autoscaling
  - Spin-up (and -down) replicas
  - Works best when services are *stateless*



- Options
  - Put all the experimental code in the engine container
  - Good for reproducibility
  - Not good for interactive science, for example when a scientist adds code on the fly in a notebook
  - All the engines need to see the same code; new engines need to catch up

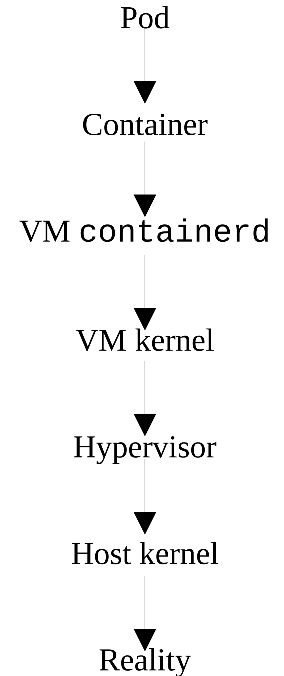
The plan (maybe)



- State lives in the broker, not the container image
  - “Initialisation queue” to grab initial setup
- Turns out this might not be needed in practice
  - A Python persistence mechanism (`cloudpickle`) handles the most common case

Issue 3: Performance tuning

- Performance can be hard to predict
  - A lot of layers between code and metal
  - Each layer has tuning parameters
  - How do these interact? Which takes precedence?
- (At least) two problems
  - Visibility of knobs and dials
  - Deciding how to twiddle them
- How is the cloud itself tuned?
  - Different settings if it expects transient jobs or prioritises throughput over performance?



Conclusion

- Getting there…
  - Quite a learning curve, lots of tooling, often poor documentation
  - ...but well worth it, for the learning and for the resulting performance and flexibility
- Lots remains to do
  - Finish the interactive science workflow
  - Instrument with proper metrics-gathering
  - Autoscale based on application-meaningful metrics
  - New structures for different sorts of computation, re-using the core code