TEXAS ADVANCED COMPUTING CENTER
Powering Discoveries That Change The World

TEXAS
The University of Texas at Austin

Exploring and Improving the Design of Abaco with TLA+

Smruti Padhy, PhD and Joe Stubbs, PhD
Cloud Computing Group, Texas Advanced Computing Center
University of Texas, Austin

Acknowledgements

- National Science Foundation - Office of Advanced Cyberinfrastructure
  - Collaborative Proposal: Frameworks: Project Tapis: Next Generation Software for Distributed Research (Award #1931439)
  - SI2-SSE: Abaco - Flexible, Scalable, and Usable Functions-As-A-Service via the Actor Model (Award #1740288)
- Abaco and Tapis Team

Talk Outline

- Introductions
- TACC and TLA+ for Cloud Software in Research Computing
- Abaco Platform
    - Introduction
    - Architecture
    - Observed Issues at Scale
- Verification Abaco with TLA+
    - Overview of the Spec
    - Invariants and Temporal Properties
- Next Steps

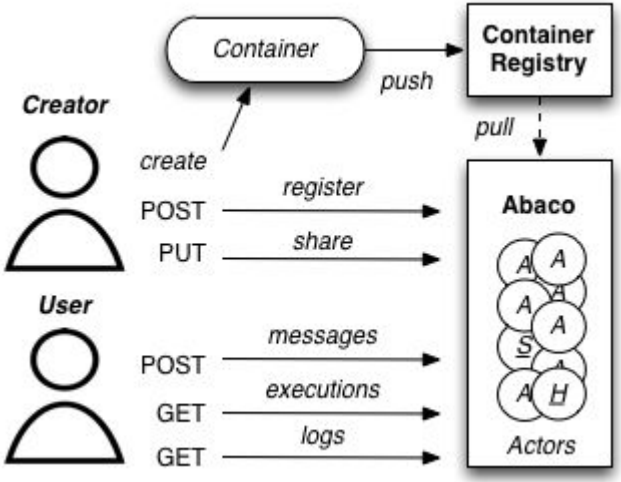
Introductions

Texas Advanced Computing Center (TACC)

Cloud Software for Research and TLA+

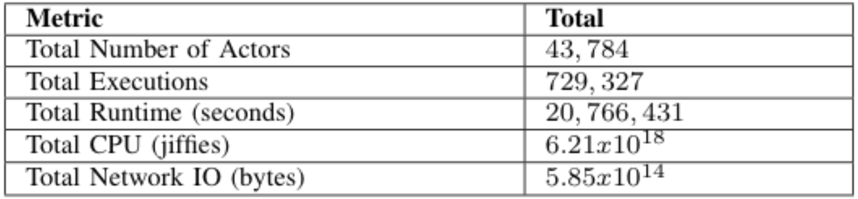
- Users are domain researchers: typically, expertise in research domain, proficient in software.
- Cloud Software lowers the barrier for researchers to use “advanced” storage and computing resources.
- Cloud Software provides programmable interfaces to resources enabling dynamic workflows.
- Research experiments depend on this software; defects in the software lead to errors in results. Contributes to the “reproducibility crisis”.
- TLA+ provides an accessible technique for applying formal methods to reduce defects.

Abaco Functions-as-a-service



- Actor Based Containers -- Linux container technology + Actor Model = FaaS platform.
- Prototype in late 2015; NSF funded in 2017.
- Now used by several research projects

Total Abaco usage since Jan, 2018

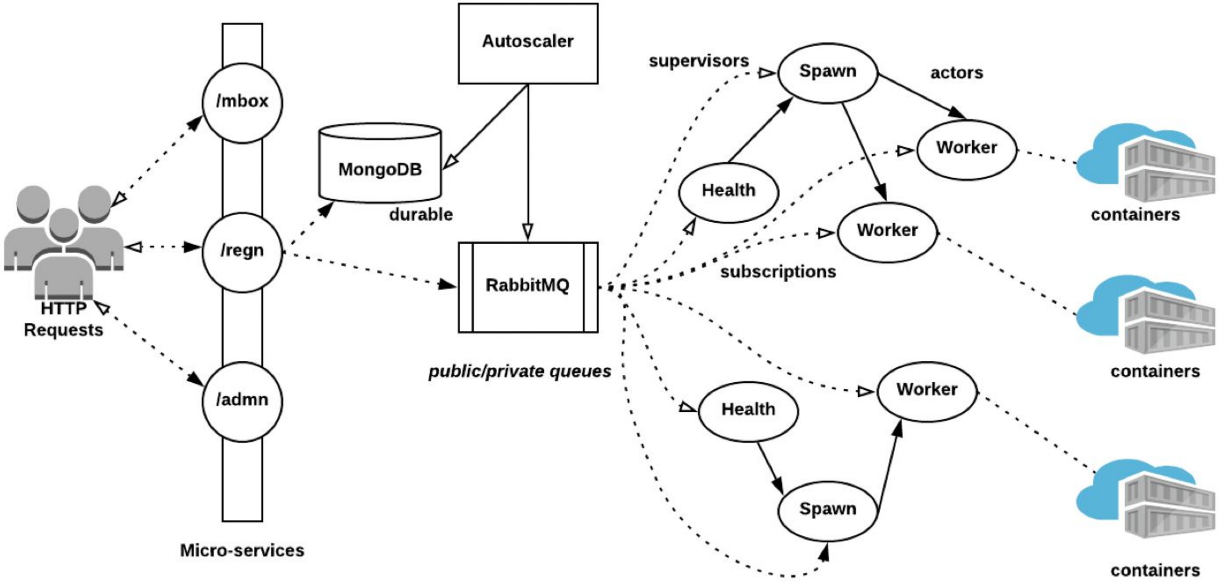


Metric	Total
Total Number of Actors	43, 784
Total Executions	729, 327
Total Runtime (seconds)	20, 766, 431
Total CPU (jiffies)	$6.21x10^{18}$
Total Network IO (bytes)	$5.85x10^{14}$

Abaco: Basic Usage

- A user defines an actor by making an API request to Abaco.
  - The request contains a reference to a publicly available Docker image.
- The user can then send messages to the actor by making an API request to URI assigned to the actor.
  - Internally, these messages get queued in the message queue assigned to each actor.
  - For each message, a docker container is launched from the actor's Docker image obtained from the registry. The system injects the original message into the container.
  - Abaco collects any results registered by the actor and the associated container logs, resource utilization, and other data, and exposes this information to the end-user through various end-points.
- Users can update actor definition (PUT), delete actors (DELETE) and share actors with other users at a specified level (READ, EXECUTE, UPDATE).

Abaco Architecture



Observed Issues at Scale

- Actor Update
  - The user makes an actor update API request with the updated Docker Image reference URL
  - Current workers must complete executions with current image.
  - For subsequent executions, including queued messages, new workers must be started with new image.
  - Abaco sends a “shutdown after completion” message to each existing worker.
    - Message sent to worker via RabbitMQ.
    - Message received by worker’s “command” thread, communication with “main” thread via shared memory.
- We observed issues where, under load, new executions were being started with a stale version of the image.
  - Upon inspection, there is a race condition -- the AutoScaler could be queuing new workers to be started at the same time the user sends a request to update the actor. There is a race between new workers getting created by Autoscaler and shutdown messages getting sent to the existing worker set by the API.

Verification of Abaco with TLA+

Spec Overview (1)

- CONSTANTS of the Spec -- serve two purposes:
  - Represent configurable aspects of Abaco itself; e.g., ScaleUpThreshold, MaxWorkersPerActor,
  - Allow us to control the size of the state space; e.g., Actors, MaxMessage, MaxWorkers, ImageVersion

- Variables of the Spec -- represent Abaco runtime state stored in persistence layers
  - MongoDB -- e.g., actorStatus \in [Actors -> { "SUBMITTED", "READY", "ERROR", "SHUTTING_DOWN","UPDATING_IMAGE","DELETED"}], workerStatus \in [Workers -> [actor:AllActors, status:WorkerState]]
  - RabbitMQ -- e.g., actor_msg_queues, command_queues

- “Top level” actions of the Spec -- represent the initial agents of change.
  - Receiving an HTTP request to a specific endpoint; e.g., ActorExecuteRecv, ActorUpdateRecv, ActorDeleteRecv,
  - Autoscaler initiating a change: i.e., CreateWorker, StartDeleteWorker.

Spec Overview (2)

Additional actions -- represent asynchronous activities triggered by top-level actions.

- Actor executions: i.e., WorkerRecv, WorkerBusy, FreeWorker
- API Requests (actor update and delete): e.g., UpdateActor, DeleteActor.
- Autoscaler commands: e.g., CompleteDeleteWorker

Finding A Problem in the Design (1)

Invariant (safety)

- TypeInvariant: All variables maintain correct type
- AllWorkersUseCorrectImageVersion - All workers of an actor will use the correct image version -- getting the right definition required new design ideas.

Temporal properties (liveness)

- AllActorMessagesProcessed - All actor messages are eventually processed
- AllCommandMessagesProcessed - All command messages are eventually processed

Weak Fairness

- For certain next actions - CreateWorker, WorkerRecv, WorkerBusy, FreeWorker, StartDeleteWorker, ...

Finding A Problem in the Design(2)

- Validated the model of spec using TLC model Checker.
- We realized quickly that, with the current implementation, AllWorkersUseCorrectImageVersion would not hold for any reasonable definition. For example:
  - “All workers use the same image as the actor” would not work.
  - “All workers use the same image as each other” would not work.
- We needed to modify the design to allow a suitable definition.

Invariants

$$\begin{aligned}
&\mathit{AllWorkersOfReadyActorsUseSameImageVersion} \triangleq \forall\, a \in \mathit{Actors} : \forall\, x,\, y \in \mathit{actorWorkers}[a] : \\
&\quad \mathit{actorStatus}[a] = \text{“READY”} \land \mathit{workerStatus}[x].\mathit{status} = \text{“IDLE”} \Rightarrow \\
&\quad \mathit{revision\_number\_for\_workers}[x] = \mathit{revision\_number\_for\_workers}[y]
\end{aligned}$$

$$\begin{aligned}
&\mathit{AllWorkersOfReadyActorsUseLatestImageVersion} \triangleq \forall\, a \in \mathit{Actors} : \forall\, x \in \mathit{actorWorkers}[a] : \\
&\quad \mathit{actorStatus}[a] = \text{“READY”} \land \mathit{workerStatus}[x].\mathit{status} = \text{“IDLE”} \Rightarrow \\
&\quad \mathit{revision\_number\_for\_workers}[x] = \mathit{revision\_number}[a]
\end{aligned}$$

Temporal Properties

$$
\begin{aligned}
&AllWorkersOfActorUseLatestImageVersion\_live \triangleq \Diamond\Box(\forall\, a \in Actors : \forall\, x \in actorWorkers[a] : \\
&\quad revision\_number\_for\_workers[x] = revision\_number[a])
\end{aligned}
$$

$$
\begin{aligned}
&AllWorkersOfActorUseSameImageVersion\_live \triangleq \Diamond\Box(\forall\, a \in Actors : \forall\, x,\ y \in actorWorkers[a] : \\
&\quad revision\_number\_for\_workers[x] = revision\_number\_for\_workers[y])
\end{aligned}
$$

$$AllActorMessagesProcessed \triangleq \Diamond\Box(\forall\, a \in Actors : Len(actor\_msg\_queues[a]) \qquad = 0)$$

$$AllCommandMessagesProcessed \triangleq \Diamond\Box(\forall\, a \in Actors : Len(command\_queues[a]) = 0)$$

Initial Outcomes: Insights and Design Changes (1)

Changes	Current	New
Actor image revision number (monotonically increasing with every update).	Autoscaler uses “image” and a flag that indicates where existing workers should be shut down.	The actor object saves the image revision on every update (ActorUpdateRecv), and workers are started with the image revision (CreateWorker).
Actor status change from UPDATING_IMAGE to READY	The first worker that is started with the new image updates the actor’s status to READY.	The autoscaler moves the actor’s status to READY (UpdateActor) only when all of its workers have the latest image.
New checks in autoscaler for creating a new worker.	Workers can be created regardless of the status of other workers.	A new worker can only be created if there are no IDLE workers (CreateWorker).

Initial Outcomes: Insights and Design Changes (2)

Changes	Current	New
New checks when deleting a worker by autoscaler.	The autoscaler does not delete stale workers and the revision is not considered; at the time the user issues an UPDATE, a shutdown message is sent to the current set of workers	A worker will be deleted anytime it is IDLE and does not have the current image revision (StartDeleteWorker).
Modify when a worker can receive a message.	A worker's main thread retrieves a new message unless the interrupt thread has communicated that it should not via shared memory.	A worker's main thread checks its revision number against the actor's revision number before retrieving a new message (WorkerRecv).

Next Steps

- Implement design changes to Abaco based on TLA+ spec.
- Look at using TLAPS for writing proofs of correctness -
    - In earlier versions of the Abaco spec, TLC checks had subtle dependencies on CONSTANTS, mostly due to suboptimal spec code, but we like the idea of removing any dependencies on constants.
    - However, need to weigh value of absolute guarantees with the time to write proofs.
- We are very encouraged by the results and plan to use it for other projects
    - Data transfers service (similar use case to that of Abaco)
    - Verifying security constraints for our “Associate Sites” feature where components of our software run in remote datacenters but coordinate via messages sent across a WAN.

Future

- We teach courses at UT Austin in the Computation Engineering Program (e.g., COE 332, Systems Design). Looking to add TLA+ to COE 332 and/or a potential sequel course.
- Looking to provide mechanism to aid and encourage researchers to utilize formal methods in their own software.

Thanks!

Spec: https://github.com/tapis-project/specifications

Abaco Project Github: https://github.com/tacc/abaco

Get in Touch:

- Smruti Padhy, spadhy@tacc.utexas.edu
- Joe Stubbs, jstubbs@tacc.utexas.edu