RAS - Resource Allowance System

Presented by Arkash Jain

Background

Business Problem

- Meta **services** like instagram, messenger, facebook need data management.
- Data **centers are across the globe** with independent MSBs (Main Switch Board)
- Business is impacted when data is lost, and why is it lost?
    - **Power Outage**
    - **Maintenance** (same cost)

Technical Problem

- **Heterogeneous hardware**: Hardware with different storage, RAM, processing speed etc. impacts data allocation
- **Latency and Locality:** Proximity of the server and network speed
- **Inefficient Resource allocation**
- **Capacity Availability**

Motivation

Capacity Allocation

Independent Failures - Servers fail, but work should not

Scalability - speed and allocation need to be balanced

Cluster Management - Different CPUs, different requirements

Workload Constraints - Issues of locality and latency for data operations.

RAS gets balance between speed and allocation efficiency

Decouple server assignment from container placement

Breaks into 2 parts Server-to-reservation assignments(continuous MIP re-evaluation), Container placement (off critical path)

RAS optimizes reservations region-wide and globally.

Adopted across facebook.

Most Common assignment via static scopes. Servers may belong to one cluster and added and removed manually

Benefit : Reduces allocations off critical path

Loss : Servers underutilized or overutilized, Manual initialization, Suboptimal

Capacity abstraction to represent set of servers dynamically assigned to logical cluster.

Capacity abstraction to represent set of servers dynamically assigned to logical cluster.

- Server to assignment off critical path.
- Container to server allocation follows.

Twine used a free server pool. When a server ran out of space, greedily task was assigned to a free one. After completion, sent to the free pool

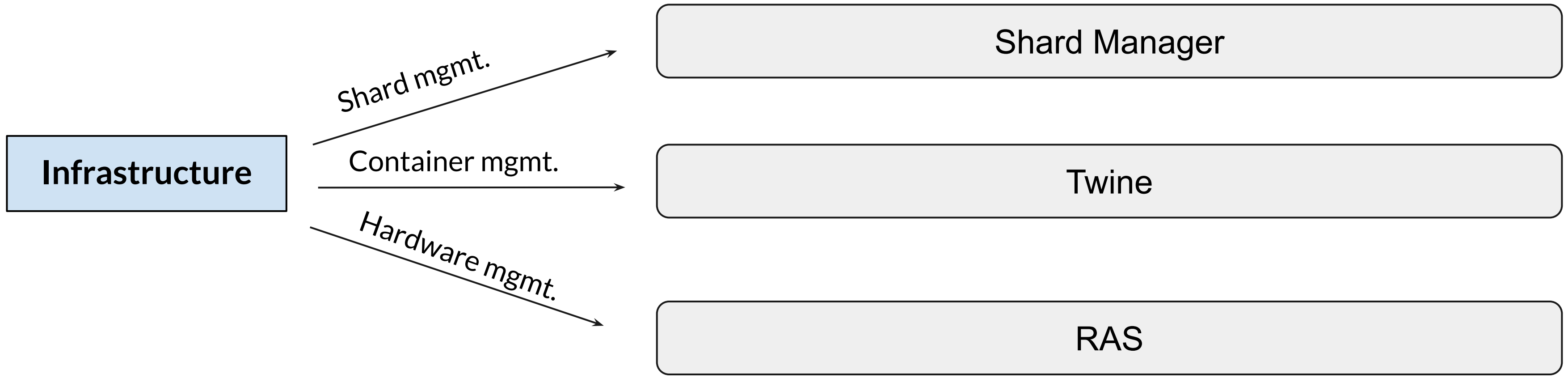
Benefit : eliminates capacity underutilization across region

Loss : Puts server-to-entitlement on critical path

2 region architecture

- **Region Level:** local optimization for regional allocation. Uses local characteristics and constraints
- **Global Level:** Overall optimization uses data center availability, heterogeneity, utilization rate within region

Structure


Infrastructure
Shard mgmt.
Container mgmt.
Hardware mgmt.
Shard Manager
Twine
RAS

Architecture

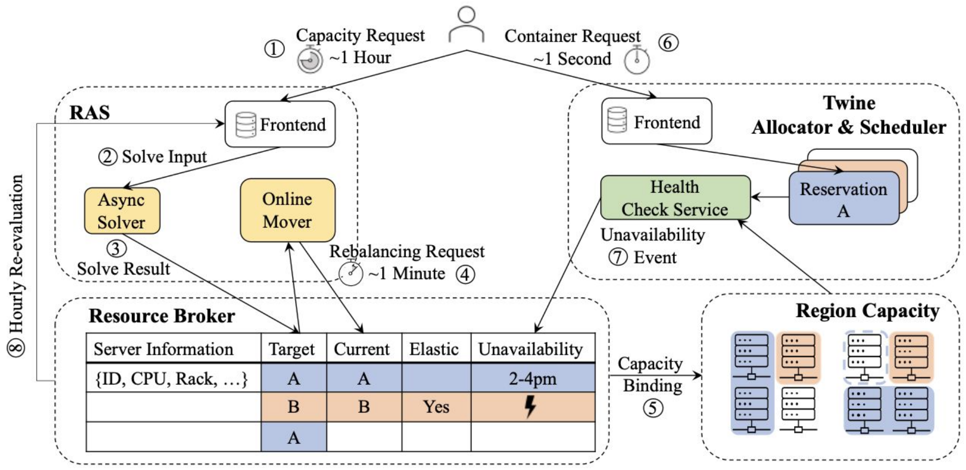
Two layered decoupled architecture**

Async solver uses MIP* to understand region capacity and do dynamic binding. The mover executes the solvers decisions.

Twine allocator performs real-time container allocation on reservations and manages lifecycle

Health Check service monitors all servers, whereas resource broker virtualizes region capacity and has reservation information.

*MIP is Mixed Integer Programming, is an optimization algorithm to allocate tenant resources using RRUs. Done every few tens of minutes
**RRUs capacity request from actual hardware



Server Information	Target	Current	Elastic	Unavailability
{ID, CPU, Rack, …}	A	A		2-4pm
	B	B	Yes	⚡
	A			

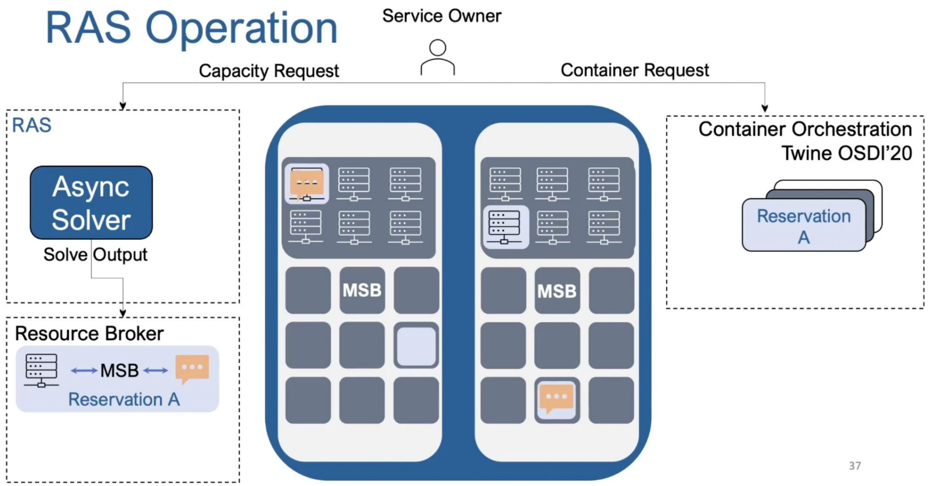
Operation

Async solver does calculations to make a solve output for resource brokers, which creates mappings to the hardware**. (This is a reservation*).

Container orchestration layer allows devs to manage resources at scale through APIs. It is used for Load Balancing, Scaling and container deployment.

When one part of the MSB fails, the MSB moves resources automatically.

*Reservation is a logical cluster that represents a materialized amount of resources Twine allocator can use.
**Hardware is abstracted through RRUs or relative resource units. They correspond to a fixed amount of resources like CPUs, memory, disk space and bandwidth. Designed to be granular, dynamic, standardized and thus flexible.



Main Ideas

Resource Management Reality

- Global Data Centers.
- Hardware Heterogeneity.
- Diverse Capacity Requests. (Explained in detail in next slide)
- Server Unavailability.

- Failure buffer sets aside capacity for maintenance and outages.
- When not in use, sent to elastic reservations in mover to optimize usage.
- For handling failure, buffer capacity is added by taking server off of elastic reservations.

Resource Management Flow

- Separated between capacity request and container request. Service owner can initiate and modify requests via capacity portal. Hardware capacity is determined.
- The async solver has a Service Level Objective (SLO) of completing each solve within one hour.
- Mapping is made between servers and reservation IDs.
- Emergency path is also made.
- Mover changes ownership using shared buffers and opportunistic capacity.
- Allocator gets reservations IDs from resource broker for optimization.

Conclusions

- Work had been done on server to cluster allocation but not much on container to server.
- RAS optimized critical path allocation. Critical path is the steps needed in the data or resource pipeline that have to be completed in a sequential order.
- RAS takes resources off the critical path by doing the following:
  - Dynamic Allocation: For more taxing operations, RAS assigns more resources to the affected stage.
  - Scheduling : Processes jobs based on their priority (if data is needed urgently, its given higher priority).
  - Optimization: Algorithms for better resource allocation.
- At the time of writing, Twine was using RAS for two years and it achieved 94% allocation for server workloads with 2% and 4% left for random failures and large scale failures respectively.
- Reduced failure rate from 15% to 4.5%.

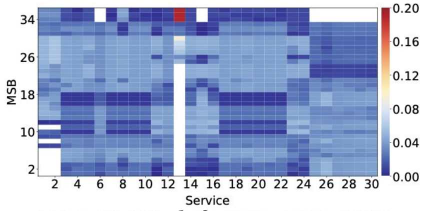


Figure 13: Spread of services across MSBs.

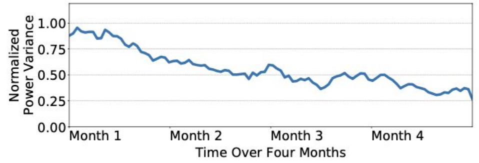
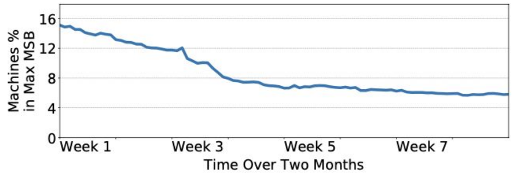


Figure 14: RAS helps reduce power usage variance across MSBs over a period of four months.

4.2 Reduce Correlated-Failure Buffers



Appendix

Resource Management Flow (Detailed Explanation)

- Service owner makes a create, edit or delete request on the capacity portal.
- Sent to the Async solver which utilizes the previous solve and current state of the region to make decisions.
- SLO is 1 hour.

- Output is mapping in the resource broker.
- There is emergency buffer capacity.
- Mover is responsible for changing the ownership of servers.
  - During emergencies, mover assigns server from shared buffer
  - Moves servers after each RAS solve output

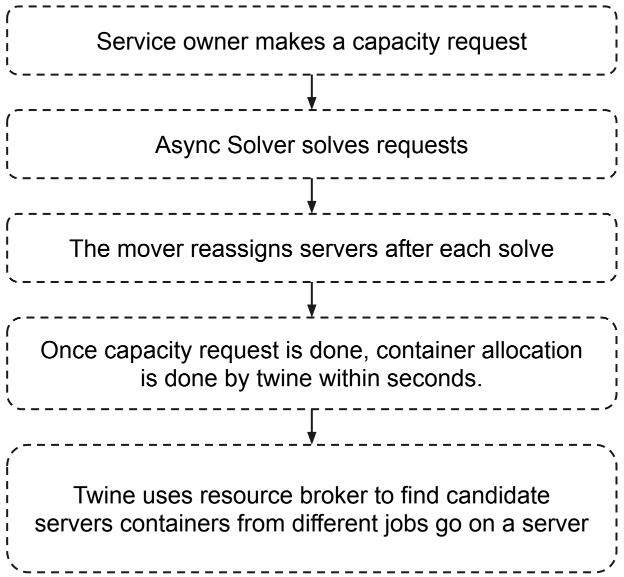
- After a capacity request is complete, the service owner can submit container requests.
- Twine's two-level architecture enables the Twine Allocator to provide swift response times of seconds on the critical path of container allocation

- Unplanned events prompt the Online Mover to provide replacement servers within one minute, and then the Twine Allocator & Scheduler move containers to those servers
- Later on, Solver provides an optimal assignment for the servers

- For correlated failure, embedded buffers are built in with no need for mover intervention. Returned in 2 phases, 75% of are returned within seconds while the remaining 25% within 30 minutes
- Its possible because the maintenance scheduling system limits concurrent maintenance operations to 25% of an MSB

- Online Mover monitors the usage of a server and changes its ownership to an elastic reservation when it is idle

Summary (cont.)


Service owner makes a capacity request
Async Solver solves requests
The mover reassigns servers after each solve
Once capacity request is done, container allocation is done by twine within seconds.
Twine uses resource broker to find candidate servers containers from different jobs go on a server

Decoupling capacity request (solver/mover) with container allocation (twine)

Mover has 3 tasks:

1) Reassigning servers after each solve
2) Reassigning servers from shared buffer in case of emergencies
3) Reassigning servers in buffers for elastic reservations

Solver runs asynchronously to generate optimal paths in each solver

Let's say a service owner needs 5 containers on a cluster to do ML on instagram images. They will generate a capacity request and twine's allocator will get a list of possible candidate servers. In the background, the solver's MIP will take the 5 containers and map them to servers under the constraints. After the solver generates a mapping, the mover will preempt containers, perform host clean up and configurations and generate a capacity binding that ties that 5 containers to the reservation.

MIP: Constraints and Objectives

Solver performs continuous optimizations to efficiently manage capacity across the data center

Constraints:

- Each placement goal can be treated as a constraint or an objective
- If a goal can block some capacity its a constraint.They are more dominant and higher priority.
- When competing constraints cannot be met, they are softened such that none regresses from its initial value and there are objectives associated with fixing as many constraints as possible.

Objectives:

- To minimize churns, RAS aims to move unused servers instead of those with running containers and strives to maintain the same move in the current solve if a move was generated in a previous solve.
- Spreads reservations across MSBs to minimize correlated-failure buffers
- Aims to reduce hotspots that may overload rack switch uplinks

- **Capacity constraint** enforces that the allocated capacity for a reservation meets its requested capacity in RRUs.
- **Availability constraint** filters out servers that are unavailable due to unplanned failures.
- **Network constraint** minimizes unnecessary cross-datacenter communication traffic by enforcing compute capacity allocated to each datacenter to match the ratio of storage.
- **Correlated- failure-buffer** constraint ensures that after losing any MSB, the reservation can still meet its capacity requirement.

MIP: Optimization

Minimize:

$$\sum_{s\in S, r\in R} M_s * \max(0, X_{s,r} - x_{s,r}) \qquad (1)$$

$$+\beta * \sum_{r\in R, G\in\Psi^K} \max\left(0, \sum_{s\in G}(V_{s,r} * x_{s,r}) - \alpha^K * C_r\right) \qquad (2)$$

$$+\beta * \sum_{r\in R, G\in\Psi^F} \max\left(0, \sum_{s\in G}(V_{s,r} * x_{s,r}) - \alpha^F * C_r\right) \qquad (3)$$

$$+\tau * \sum_{r\in R} \max_{G\in\Psi^F}\left(\sum_{s\in G} V_{s,r} * x_{s,r}\right) \qquad (4)$$

Subject to:

$$\sum_{r\in R} x_{s,r} \leq 1, \qquad \forall s \in S \qquad (5)$$

$$\sum_{s\in S}(V_{s,r} * x_{s,r}) - \max_{G\in\Psi^F}\left(\sum_{s\in G} V_{s,r} * x_{s,r}\right) \geq C_r, \qquad \forall r \in R \qquad (6)$$

$$\left|\frac{\sum_{s\in G}(V_{s,r} * x_{s,r})}{C_r} - A_{r,G}\right| \leq \theta, \qquad \forall r \in R, G \in \Psi^D \qquad (7)$$

- **Stability objective (exp1)** ensures servers are not moved out of reservations too frequently. A cost is imposed on each server that is moved out of a reservation. (higher cost = servers with active running containers)
- **Spread-wide objective (exp 2, 3)** ensures capacity is spread out at the rack and MSB (multi-service building) levels. The parameters α K and α F set a threshold for the maximum proportion of capacity that can be allocated within a single physical scope, and β is the penalty associated with servers exceeding this threshold.
- **Assignment variables (exp 5)** represents the basic assignment constraints used throughout the problem. It ensures that each container is assigned to a server and that each server is assigned to a rack and an MSB.
- **Embedded correlated-failure buffer (exp 6)** ensures a reservation has enough remaining capacity after the failure of any MSB, while Expression 4 minimizes the correlated-failure buffer.
- **Network affinity constraints (exp 7)** enforces a reservation's preference for physical data centers. The values of Ar,G are determined outside of RAS and dictate the amount of capacity that should be allocated from different datacenters for a reservation.

MIP: Explaining objectives and Constraints

- **Stability objective (exp1)** ensures that servers are not moved out of reservations too frequently, as moving servers around too much can cause instability and disrupt running containers
- **Spread-wide objective (exp 2, 3)** helps to prevent power and network hotspots and distribute capacity more evenly.
- **Assignment variables (exp 5)** ensures that each container is assigned to a server and that each server is assigned to a rack and an MSB.
- **Embedded correlated-failure buffer (exp 6)** aims to prevent the failure of a single MSB from causing a reservation to fail as well.
- **Network affinity constraints (exp 7)** if a service's data resides in a datacenter, its compute servers should also come from that datacenter to minimize cross-datacenter traffic. This objective aims to optimize network traffic and reduce latency.

MIP: Solving

Exploiting Symmetry:

- Express the basic MIP model with assignment variables $x_{s,r}$ which are 1 when server s is given to reservation r and 0 otherwise. In the MIP formulation, there are large groups of servers where all of their assignment variables $x_{s,r}$ have the same coefficients in all constraints and objectives
- In practice, these groups of servers are identical in terms of our modeling. Thus, they are merged into a single integer variable representing how many of that equivalence class are assigned to a particular reservation

Phased Solving

- In the first phase, RAS solves the problem without any rack-related goals, which allows grouping more symmetric servers into one assignment variable and reduces the problem to less than ten million variables.
- In the second phase, RAS solves the problem with all goals in phase one plus rack goals, but limits the problem to a subset of reservations in order to keep the number of variables under a limit
- The reservations that have the worst rack-level objectives are prioritized to be selected in this second phase.

MIP: Phases

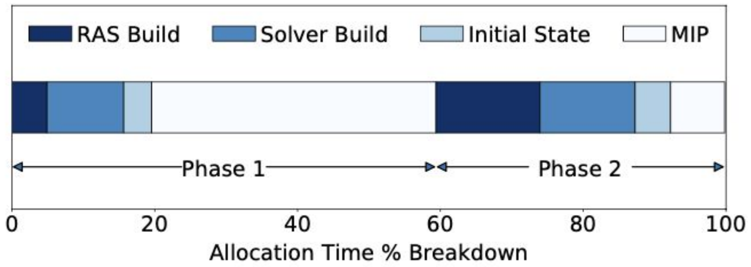


Figure 8: RAS allocation time breakdown.

- Phase 1 **accounts for 60% of the total allocation time**. Each phase is broken down into four steps. The RAS Build step builds the objectives and constraints required by RAS. The Solver Build step builds the constraints and objectives based on the requirements and applies the symmetric-server optimization. The Initial State step provides to the solver the initial assignment and perform the initial LP solve. Finally, the MIP step does the actual MIP solving

- **Phase 1 spends 67% of its time in the MIP step**. By contrast, **Phase 2 spends only 19% of its time in the MIP step, whereas almost 70% of its time is split equally between the two build steps.** These differences are due to differences in complexity between the two phases. Phase 1 performs a coarse-grained solve and takes into account the region's entire capacity and ensures basic capacity for reservations and failures buffers. Phase 2 further refines the server assignments done by Phase 1.

Challenges

- The RAS solver may take up to one hour to grant a new capacity request, which is too slow if the capacity is needed to handle an urgent site outage.
- For more than one MSB outage, RAS considers all reservations equally and impacts allocation
- If failure is above tolerant limit, rigid capacity is problematic
- Continuous optimization increases preemption rates. Owners need to be more flexible and re-evaluate how to spread shards across capacity