Resource Management in Aurora Serverless

Paper by Amazon Web Services

Authors - Bradley Barnhart, Marc Brooker, Daniil Chinenkov, Tony Hooper, Jihoun Im, Prakash Chandra Jha, Tim Kraska, Ashok Kurakula, Alexey Kuznetsov, Grant McAlister, Arjun Muthukrishnan, Aravinthan Narayanan, Douglas Terry, Bhuvan Urgaonkar, Jiaming Yan

Presented by Avinash S Atluru, Aditya Chaurasia, Chang Che, Jinghao Miao, Wesley Gao

Georgia Tech

Background & Motivation

Introduction to Aurora Serverless

- **Definition**: An on-demand, autoscaling relational database service with MySQL and PostgreSQL compatibility.
- **Primary Benefit**: Eliminates the need for customers to manage database capacity, reducing both costs and complexity.
- **Capacity Management**: Utilizes Aurora Capacity Units (ACUs), allowing dynamic scaling between user-defined minimum and maximum limits.

Problem and Motivation

- **Traditional Database Scaling Issues**
  - Users must provision fixed resources, leading to inefficiencies.
  - Over-provisioning during low demand, and performance issues at peak times.

- **User Demand**: Growing need for flexible, cost-efficient, and self-managing databases.

Why Aurora Serverless?

- **Aurora Serverless Solution**
  - Scales resources up and down as needed, reducing costs and ensuring performance stability across workload variations.

- **Scaling Approach**:
  - Granular Scaling: Adjusts in small ACU increments.
  - Usage-Based Charging: Pay only for what is used.
  - Efficiency and Availability: System ensures high host utilization and fast resource adjustments to handle workload surges.

Related Work

Aurora VS other database services

- **Heracles, Sharc, Pythia**
  - Use Approaches like “colocating CPU- vs. memory- vs. network-intensive workloads” to optimize resource allocation; however, they often lack the needed for real-time adaptation.
- **Modellus**
  - Use approaches like queueing theory and control methods to optimize resources. Aurora Serverless uses recent demand data rather than long-term predictions, which achieves better simplicity and accuracy.
- **Remus, Sandpiper**
  - Live migration as a method for resource management has been discussed in these research, but its complexity has limited its practical deployment. Aurora Serverless overcomes these limitations, providing a scalable solution that adjusts resources flexibly based on demand.

Overview

What is Aurora Serverless Accomplishing?

- **Fleet-Wide Management:**
  - Ensures balanced resource utilization with host allocations and migrations
- **Host Level Management:**
  - Manages resources for individual instances on a single host
- **In-Place Scaling:**
  - Dynamically adjusts instance resource allocations (ACUs) to meet changing demands without downtime.
- **Boundary Management:**
  - Ensures efficient resource usage by adjusting reserved ACUs based on observed trends
- **Regulated Scale-Up**:
  - A **token bucket mechanism** controls the rate of instance scaling.
  - Prevents fast-growing instances from saturating hosts, ensuring smooth resource allocation and allowing time for live migrations.

The Aurora Serverless Capacity Bounds

Aurora Serverless Capacity Bounds

- Aurora Serverless dynamically adjusts the necessary resources (such as CPU, memory, and throughput) to match demand
- Resources are measured by Aurora Capacity Units (**ACUs**)
    - Combination of 2GB of memory, corresponding CPU (0.25 vCPU), networking, and storage throughput
- Capacity Bounds (the range)
    - Minimum of 0.5 ACUs
    - Maximum of 128 ACUs
- **Goal**: Ensuring consistent performance, cost-efficiency, and responsiveness

Aurora Serverless Capacity Bounds: ACUs and Scaling

- Given a boundary range, ACUs are scaled up or down based on the demand from the client
- Scaled how?
  - Granularity - Aurora Serverless adjusts the capacity in larger steps instead of tiny increments
    - Helps avoid frequent adjustments that may be costly to the client, optimizing cost and performance stability
- Example:
  - If an increased workload is experienced, then AS will not automatically increase the ACUs until a certain threshold is reached
- Customer charges are calculated at 1-second granularity, offering a pay-as-you-go experience

Aurora Serverless Capacity Bounds: Design Factors

- Main factors that were considered by the authors when designing AS were:
  1. **Pay-as-you-go**
     a. how close to a fully pay-as you-go experience can we offer the customer?
  2. **Quick resume**
     a. how efficiently and quickly can we resume a customer that returns after a period of inactivity?
  3. **Utilization**
     a. at how high a utilization level can we operate our infrastructure?'
- Trade-offs:
  - Fully pausing databases save costs but slows resumption
  - Setting a minimum ACU allows for cost savings while also being able to quickly resume

From ASv1 to ASv2

ASv1 ?

Session Transfer	
Relaunch Needed	
Find Quiet Point	Influence User Experience
Session NOT supported!	Temporary Tables
Features NOT added!	
Only 2x or 0.5x scaling	Cost-Efficient OR Fast Response

ASv2 ?

Scale IN-PLACE	
Relaunch Not Needed	
Scale Simultaneously	Can’t feel it
No Session Transfer	
No Feature Difference	
Scale by ±0.5 ACUs	Cost-Efficient AND Fast Response

ASv2 !

Memory/CPU Hot (un)plug

Live Migration of Instance

Virtual Machine Arch

Scalability

ASv1 really helped!

Fleet Wide Resource Management

Live Migration-Based Dynamic Instance Re-Packing

- **Heat Metrics**:
  - Hosts are monitored for resource usage along multiple dimensions: **CPU, memory, network bandwidth, and I/O**.
  - A host is flagged as hot if its aggregate reserved ACUs exceed a predefined threshold
- **3-Stage Heuristics for Instance Selection**:
  - **Stage 1: Filtering**:
    - Exclude instances unsuitable for migration (e.g., those recently migrated)
  - **Stage 2: Ranking**
    - Score instances based on resource usage and migration cost, prioritizing high-impact migrations.
  - **Stage 3: Selection**
    - Choose the instance that balances heat reduction and migration efficiency.
    - Uses two scores: one relative to the ACUs and one that linearly aggregates the resource metrics

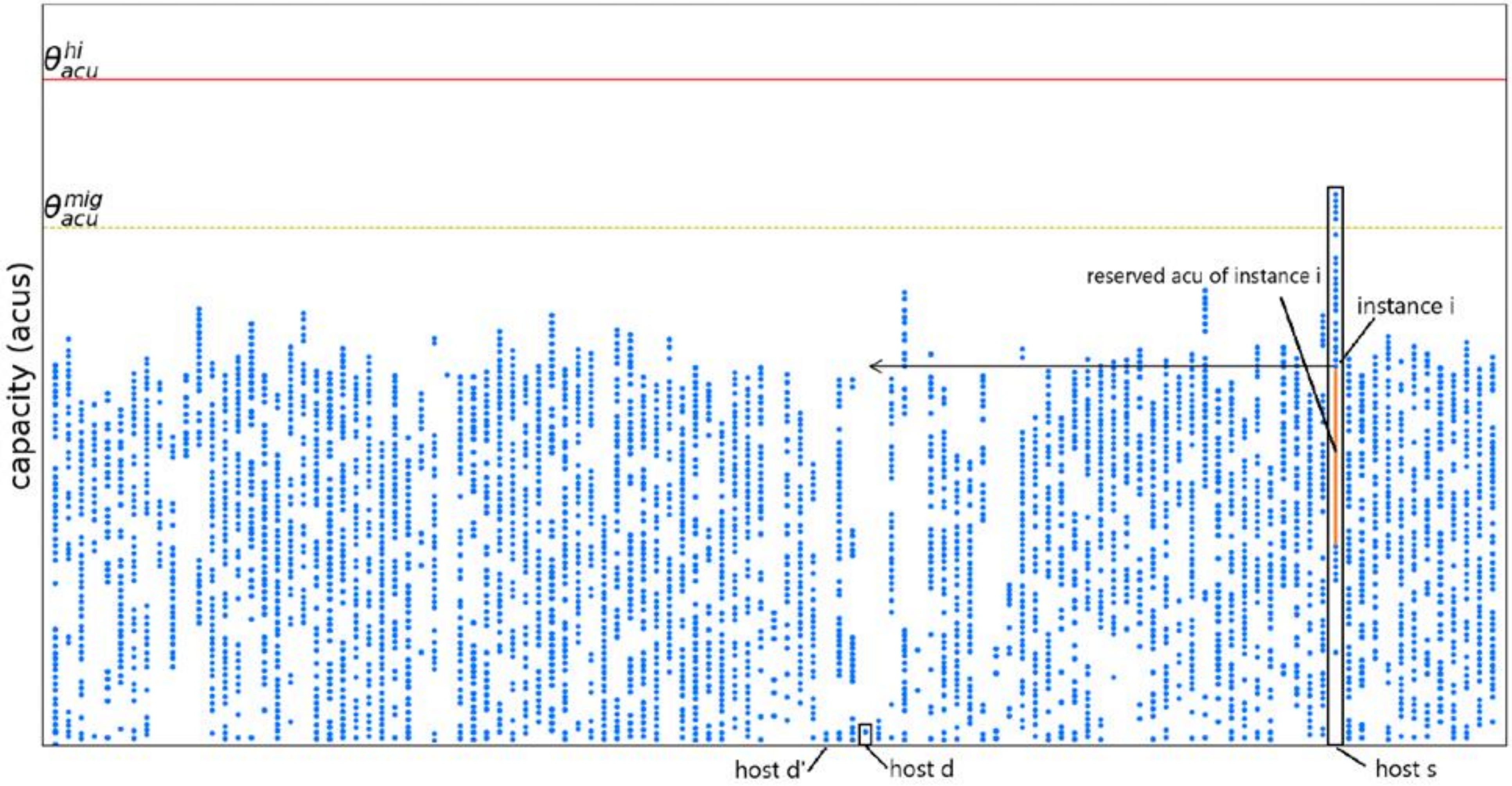
Georgia Tech

Live Migration-Based Dynamic Instance Re-Packing

- **Destination Host Selection**:
  - **Filters**:
    - Ensure the host has sufficient capacity and supports migration.
  - **Ranking**:
    - Prioritize hosts with minimal failures and better resource balance.
  - **Scoring**:
    - Optimize load balancing across resources in host (CPU, memory, etc.).
      - First score determines the heat on the host after adding the instance
      - Second score determines the overall balance of resources on the host
- **Unbalanced Load Strategy**:
  - The protocol intentionally aims for unevenly distribution among the hosts such so some hosts have enough headroom for serving as live migration destinations

Georgia Tech

Example



Avinash
Atluru
Georgia Tech

New Instance Placement

- Follows same three step process to choose the host for the new instance
- Problem the team faced was determining the resource needs of the new instance without having much knowledge on the instance itself
    - Aurora automatically chooses the minimum amount of resources as specified by the customer's min and max thresholds
    - The system will automatically scale from the minimum if the instance requires additional resources
    - Underestimating was seen to be better than overestimating since Aurora scales up faster than it scales down preventing wasted resources

Georgia Tech

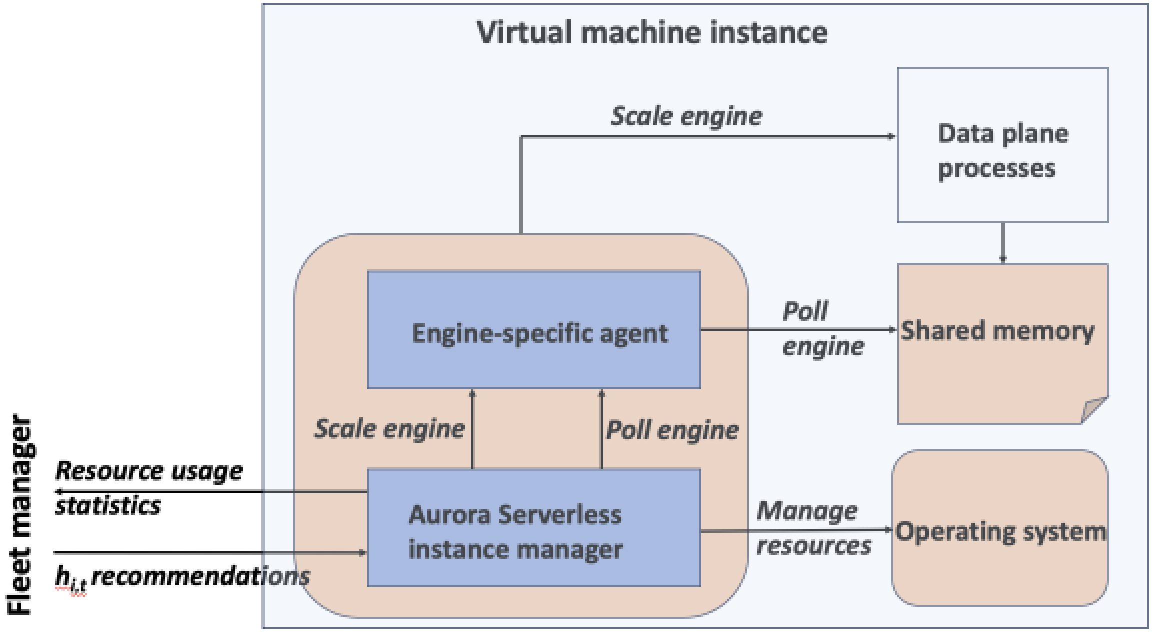
Fleet Size Adjustment

- **Demand Prediction and Threshold-Based Scaling**:
  - The fleet manager employs fleet-level demand prediction to trigger additional hosts
- **Threshold Levels:**
  - The system triggers additional procurement upon a fleet utilization exceeding a predetermined threshold.
- **Fleet Size Limitations**
  - Larger fleets lead to **higher overhead** for data collection, processing, and decision-making, which can affect system performance and scalability.
  - Fleet size is deliberately kept below a threshold that allows the entire fleet's health to be monitored and computed using a **single heat management server**.

Resource Management within Host

Instance Manager

- A library encapsulating serverless resource management functionality.
- One per instance
- Manages:
  - **Data collection**: Monitors engine-specific resource usage (e.g., buffer pool size, memory, CPU).
  - **Scaling policies**: Dynamic in-place scaling and boundary management.
  - **Resource limits**: Enforces scaling boundaries using mechanisms like cgroups and resource on/offlining.



Aditya Chaurasia

Georgia Tech

Enabling Mechanisms

- **Data Collection**
  - Collects engine and OS metrics every second for fine-grained responsiveness.
  - Buffer pool size estimate - Estimated by engine
  - Other usage statistics - Guest OS

- **Virtualization**
  - Instances run in secure VMs using the Nitro system for low IO latency and scalable CPU/memory provisioning.
  - Provides strong isolation between instances for security.

Enabling Mechanisms - continued

- **Efficient Memory Scale-Up**
  - Collects engine and OS metrics every second for fine-grained responsiveness.
  - Key Mechanisms
    - Memory Offlining: Dynamically releases memory back to the host.
    - Cold Page Identification: Frees or swaps out infrequently used pages.
    - Free Page Reporting: Reports 2MB free blocks for hypervisor reclamation.
    - Compaction: Coalesces 4KB free pages into 2MB blocks for efficiency.

- **Boundary enforcement**
  - Ensures instance is allocated resource based on “boundary” established by scaling policies.
  - 2 mechanisms to manage instance CPU/memory allocations
    - Cgroups: Enforces precise CPU and memory quotas.
    - CPU/Memory On-Offlining:
      - Adds/removes vCPUs or memory to handle spikes.
      - Reclaims unused memory efficiently (2MB blocks).

Policies

- **Boundary Management**
  - Dynamically adjusts the resource allocation boundary based on its **recent** usage patterns
  - Ensures reserved ACU stays slightly above current usage for quick scaling.
  - Key Considerations
    - Agile Growth Detection
      - Monitors memory, CPU, network, and IO every second.
      - Allocates more resources if current usage exceeds limits, up to the customer-defined maximum.
    - Regulated Growth
      - Controlled scale-up rate to avoid overwhelming hosts and enable live migration.
      - Scale-down is cautious to prepare for potential workload spikes.
      - Token Bucket system employed

Policies - continued

- **In-place scaling**
  - Provides rapid scale-up without disruption, ensuring sufficient resources are allocated for growing demands.
  - Employs conservative scaling down to avoid premature resource release.
  - Process
    - **Deciders**: Assess resource-specific needs (e.g., memory, CPU, network, storage).
    - **Combining Deciders**: Single projection derived by taking the **maximum** need across all deciders.

Empirical Observations and Evaluation

Datasets and metrics of interest

- Authors analyzed data from two fleets over different time periods:
  - Fleet 1 in us-east-1 and Fleet 2 in us-west-2
  - Both fleets used real-world observations and simulations
- Wanted to measure two key metrics:
  - **Operational efficiency**: focuses on how well servers are utilized
  - **Customer experience**: measures how elastic and responsive the system is when scaling
- Specific metrics given:
  - Scale up events satisfied in-place vs. via live migration
    - Fewer migrations is indicative of better placement strategies
  - Hosts that are deemed “hot” during scale-ups
    - For hot hosts, their max ACU is temporarily limited to avoid overloading
  - Impact on workload due to remedial actions

Georgia Tech

Customer experience observations

- Fleet 1
  - 16,440,024 scale-up events across 33,792 total instances
  - Live migrations: only 2,923 scale-up events needed one or more live migrations
  - Single migrations: 52% of those needing migrations only required one
  - 198 cases of hot hosts breaches
- Fleet 2
  - 8,151,229 scale-up events across 12,467 total instances
  - Live migrations: only 1,214 scale-up events needed one or more live migrations
  - Single migrations: 55% of those needing migrations only required one
  - 48 cases of hot hosts breaches
- Observations between the two fleets show that Aurora Serverless repacking and placement strategies are effective

Comparison against an alternative re-packing strategy

- Baseline Method:
  - Concentrate instances on fewer hosts.
  - Increases host utilization but limits spare capacity for migration.
  - Requires more migrations to address heat.

- Aurora Serverless Strategy
  - Deliberately leaves some hosts lightly loaded.
  - Reduces migration frequency.

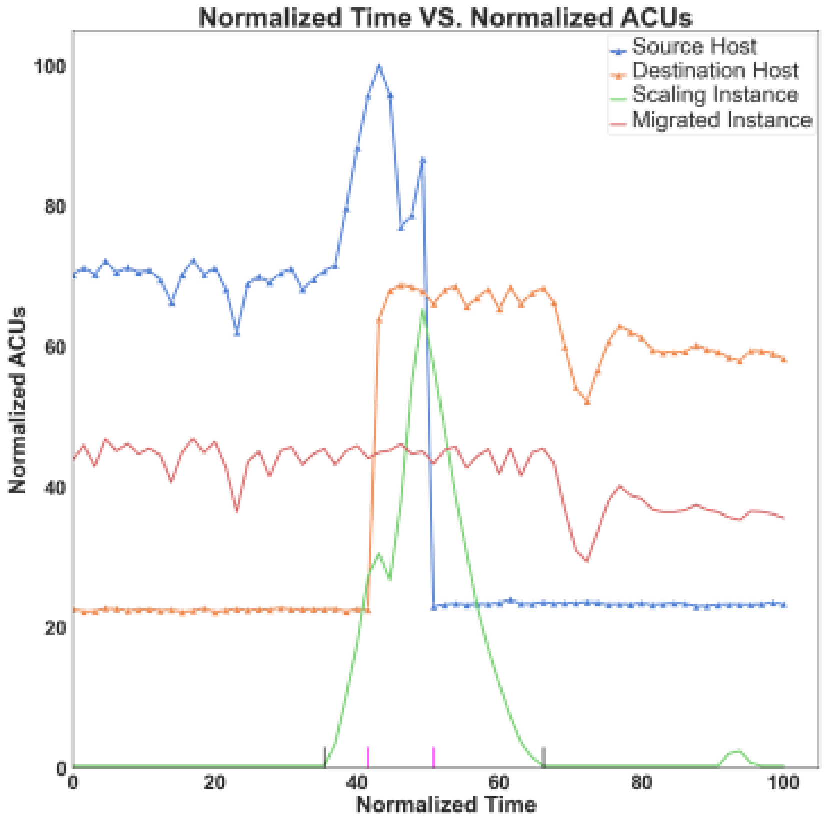
- Comparison
  - Aurora Serverless requires 82% fewer migrations in Fleet 1 and 57% fewer migrations in Fleet 2 compared to the baseline.
  - Aurora Serverless requires 10% fewer utilization of hosts in Fleet 1 and 12% in Fleet 2, increasing system flexibility.

A close look at a migration-assisted scale up

- An instance that scaling up was satisfied in-place

- Timeline
  - Heat scaling up at around 35 time units
  - Reach the threshold at around 41 time units, the migration starts.
  - Migration ends around 50 time units

- Observation
  - The live migration is efficient and ensures resource availability without significant performance drops.



Lessons & Takeaways

Lessons and Takeaways

Start Simplest

ONLY add based on needs

Reactive ✅ Predictive ❎

Fleet-wide + Host-level

Specialized OS kernels

Georgia Tech

Conclusions

Conclusions

Token Buckets: Controllable!

Reactive!

Fleet-wide + Host-level

Live Migration

About future…

Georgia Tech

About future…

Predictive!

+ Reactive!

Resource Combination

Machine Learning…

Study Questions

How does Aurora Serverless dynamically manage resource allocation within a host while ensuring predictable elasticity and minimal resource contention?

What trade-offs are involved in balancing high host utilization with seamless scale-up in Aurora Serverless, and how are these trade-offs addressed by mechanisms like live migration and regulated growth?

Georgia Tech