Understanding YARN and HAWQ Resource Mgmt

Apache HAWQ

Dan Baskette
Director of Technical Marketing

Pivotal

Agenda

Basics of Hadoop YARN
HAWQ & YARN integration
Demo
Q&A

Pivotal

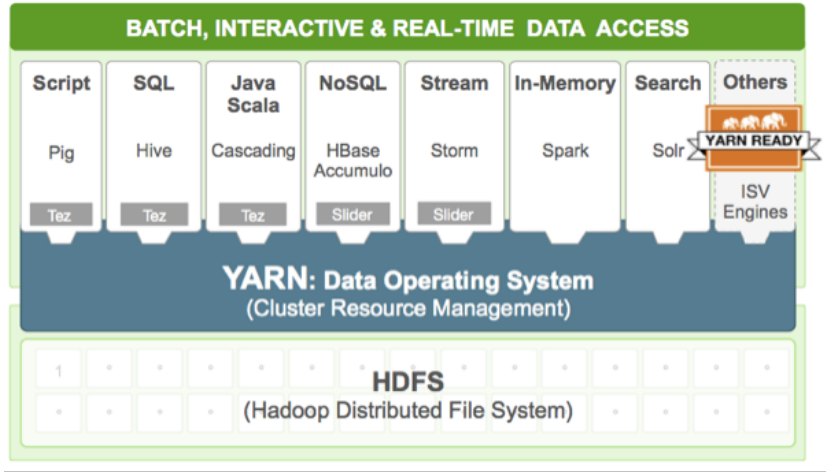
What is YARN

- YARN Basics
- Why do you care?
- YARN Components and Architecture

Pivotal

YARN 101

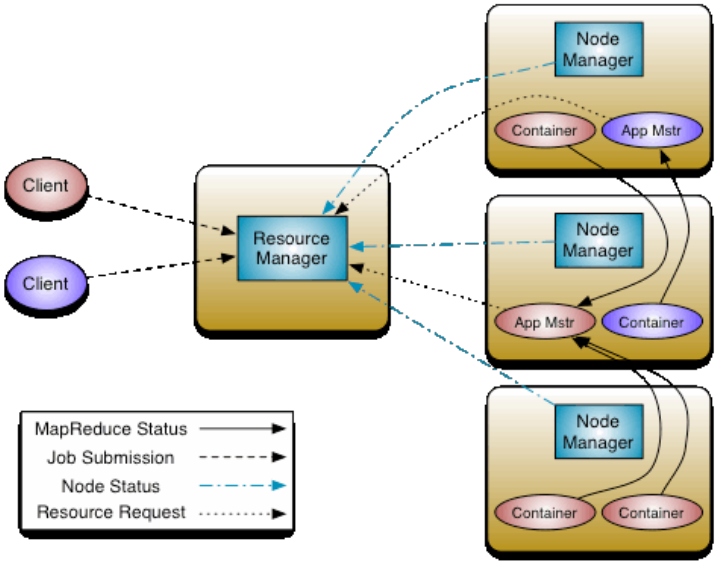
- Why YARN? (limitation of mapreduce 1)
  - Multi-tenancy (spark, hbase,tez,accumulo,hawq...)
  - Resource utilization (SLA)
  - Scalability (throttling resource dynamically)
  - Compatibility
- Cluster resource management
  - Global resource allocation
  - Local resource assignment and reporting
- What resources (resource model)
  - Heterogeneous servers/racks
  - CPUs
  - Memory
  - Network, disk IO……(on the roadmap)



Pivotal

How Yarn works

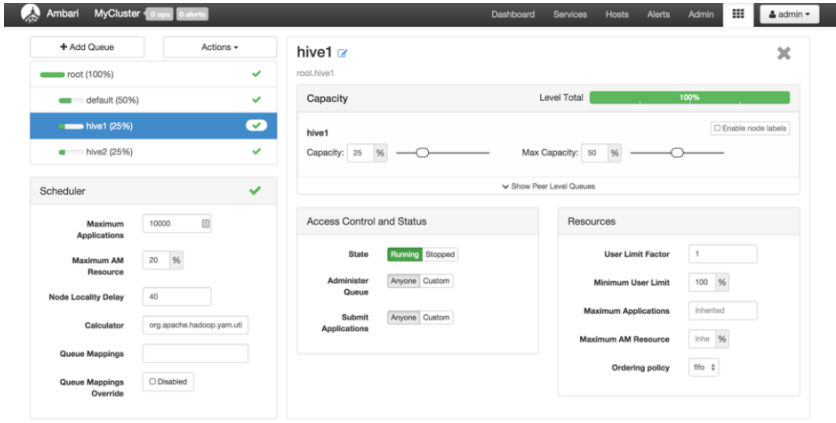
- Yarn components
  - Resource manager: master for managing **global** resources
  - Node managers: master for **local** resource allocations.
  - Application master: per **application**, manages application life cycle and task scheduling.
  - Container: abstracted set of **resources** (cpu core/memory) granted by resource manager to run a task of application.
- Yarn application life cycle

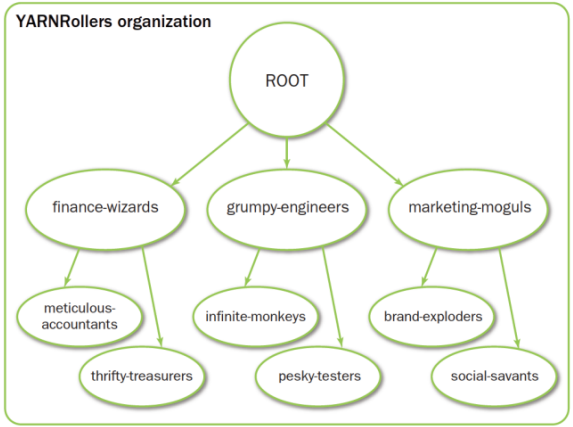


Pivotal

Capacity scheduler and queues

- Capacity scheduler
  - Elasticity with Multi-tenancy
  - Security
  - Resource awareness
  - Granular scheduling
  - Locality
  - Scheduling policies
- Queues
  - Hierarchical
    - Parent-queues don't accept applications.
    - Leaf-queues have no child queue and accept applications.
  - Access controls





Pivotal

Apache HAWQ Resource Management and Integration with YARN

Pivotal

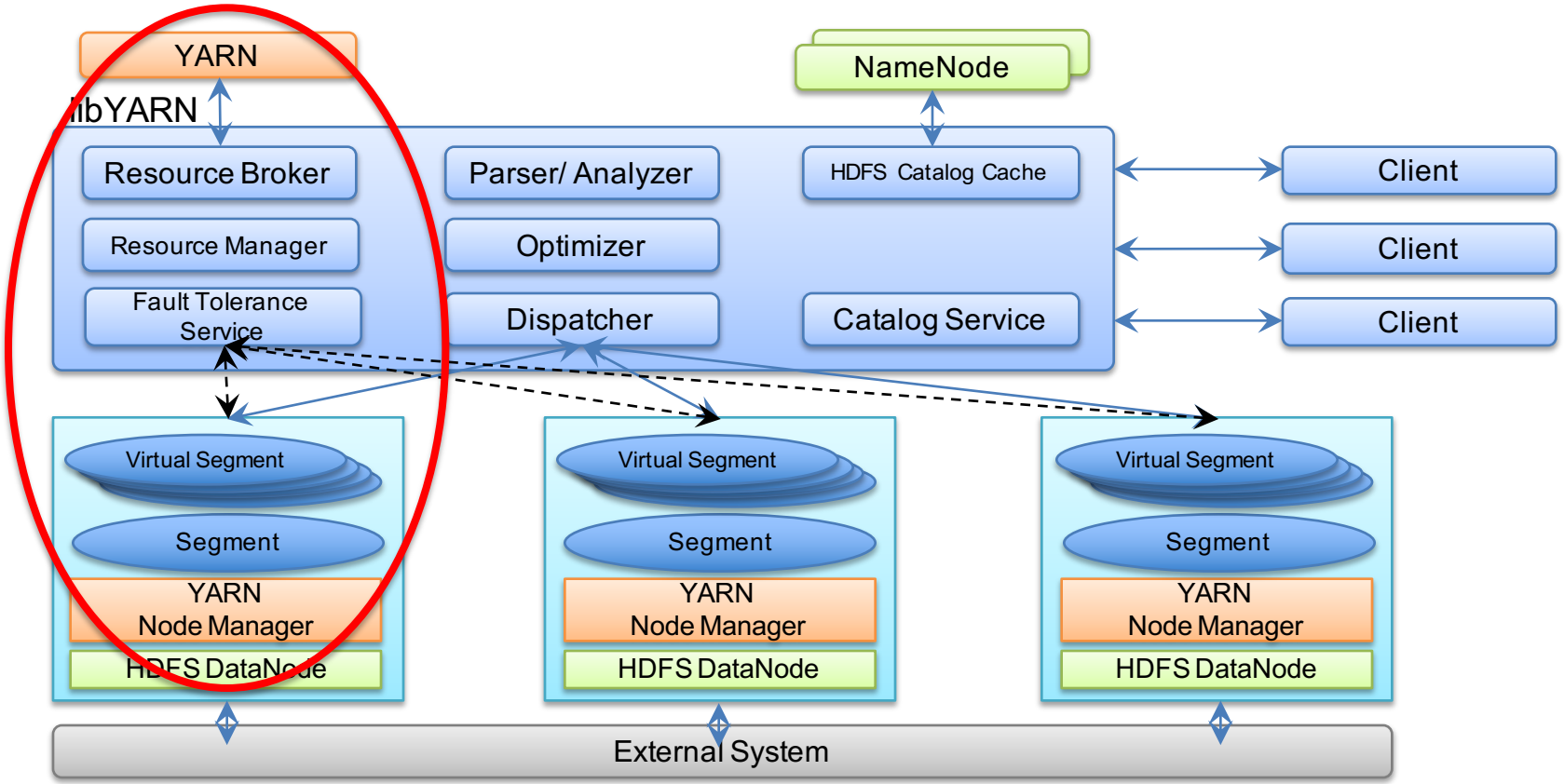
Resource Management in HAWQ

3-Tier Resource Management

Global (YARN)	HAWQ (Resource Qs)	Query (Internal)
Cluster level	HAWQ Internal	Query level
Cluster-Admin defined	HAWQ-Admin defined	System defined
Hardware efficiency	Multi-tenancy	Query Optimization
Share with MR/Hive/+	Workload prioritization	Operator prioritization
Defined in XML	Defined in DDL	Dynamic

Pivotal

Apache HAWQ High-Level Architecture



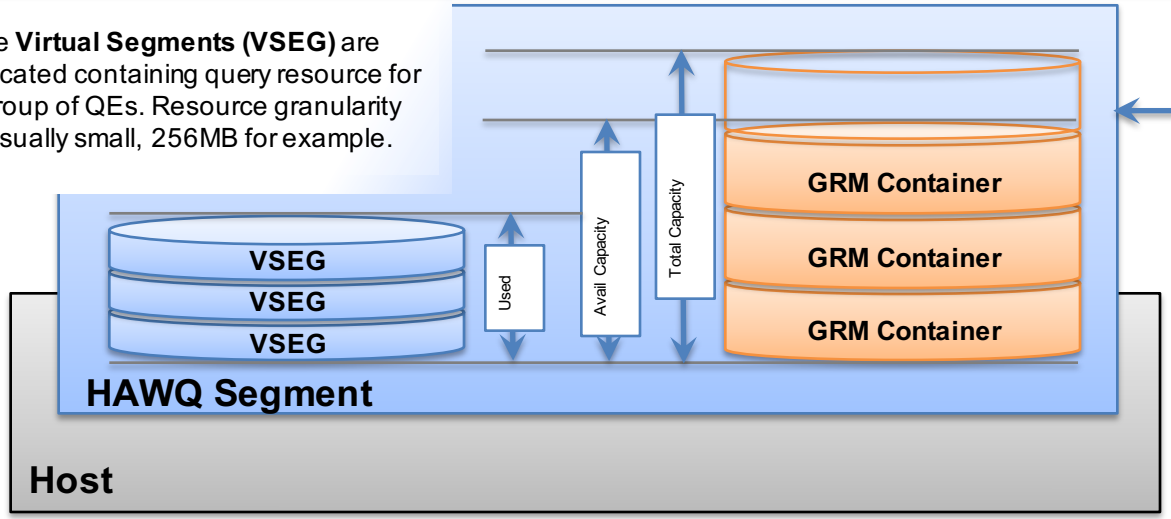
Pivotal

Apache HAWQ Resource Concepts

Small granularity, High frequency resource negotiation between QD and Apache HAWQ RM.

Large granularity, Low frequency resource negotiation between Apache HAWQ and GRM server.

One **Virtual Segments (VSEG)** are allocated containing query resource for a group of QEs. Resource granularity is usually small, 256MB for example.



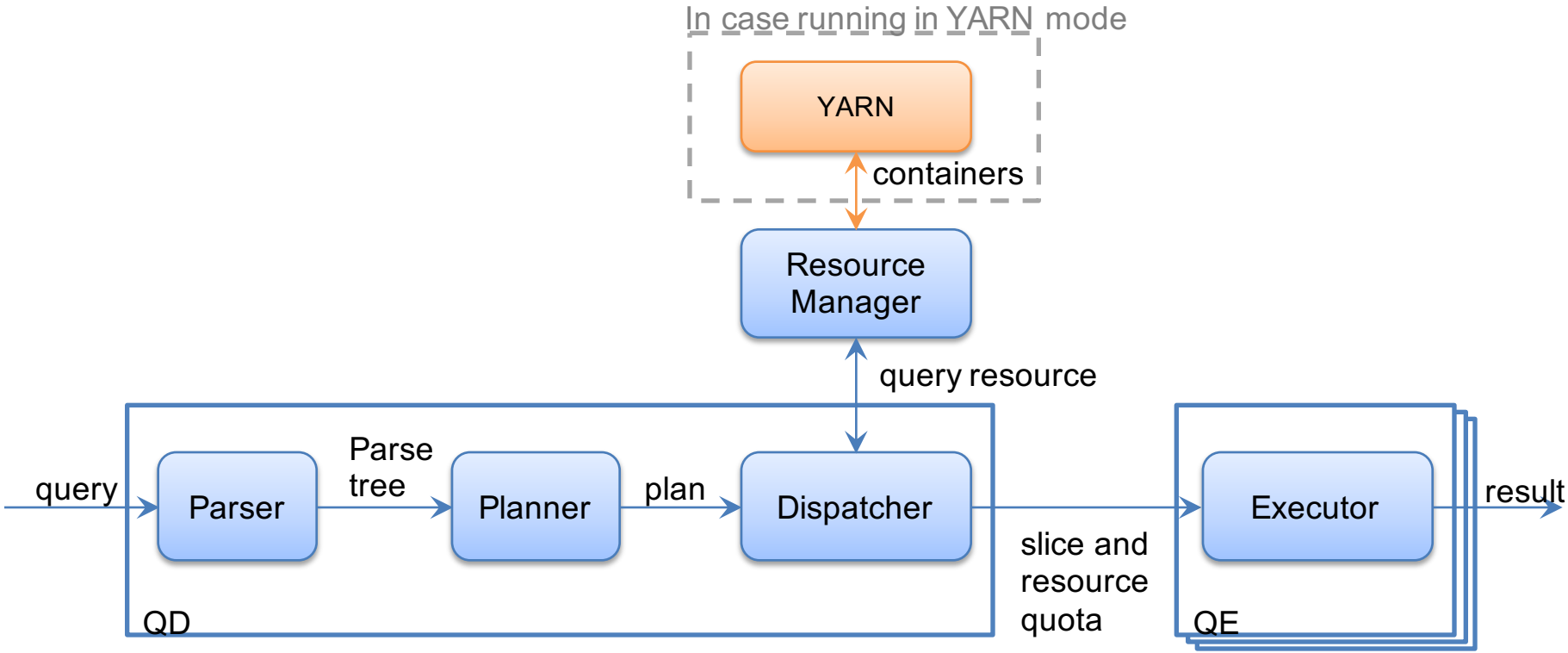
Global resource manager (GRM) containers are assigned to one **HAWQ segment**. Resource granularity is usually as large as GBs per container. YARN is considered as one GRM.

HAWQ Segment manages the resource consumable for QEs.

One host has at most one segment deployed for one HAWQ cluster.

Pivotal

High-Level Flow of Query Execution



Apache HAWQ in YARN Mode

- Apache HAWQ forks resource broker process to register itself as a YARN application to YARN Resource Manager.
- Resource broker process acts as an ***unmanaged application master***. An unmanaged application master is managed outside of YARN resource manager. There isn't a container launched for application master.
- A thread in resource broker process is created sending heartbeat to YARN resource manager on behalf of Apache HAWQ to keep registered application active and alive.
- QEs are running on segments with Node Manager running. If a HAWQ segment has no Node Manager running, it will be marked as DOWN by HAWQ and will not be used by resource pool.
- Apache HAWQ activates YARN containers by dummy processes once getting the allocated container list.
- Apache HAWQ enforces resource usage in all segments by itself.

Configure to Integrate Apache HAWQ with YARN

- YARN configuration required by Apache HAWQ
  - Capacity Scheduler is required
  - Assign one queue exclusively to Apache HAWQ

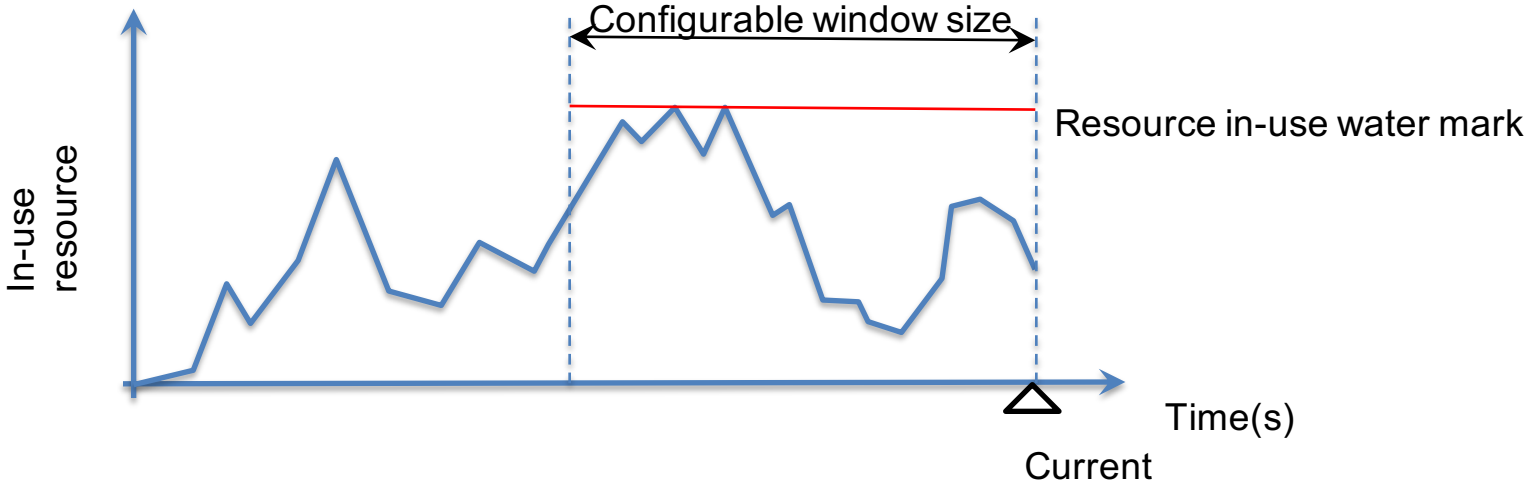
Property	Definition
hawq_global_rm_type	Apache HAWQ global resource manager type that indicates Apache HAWQ runs in corresponding modes. 'yarn' means enabling yarn mode.
hawq_rm_yarn_address	YARN resource manager address including.
hawq_rm_yarn_scheduler_address	YARN scheduler address.
hawq_rm_yarn_queue_name	YARN queue name that used by Apache HAWQ resource manager.
hawq_rm_yarn_app_name	Apache HAWQ YARN application name.

Pivotal

Friendly Apache HAWQ YARN Resource Return

Apache HAWQ resource manager friendly uses YARN allocated resource, which makes it flexibly co-exist with another YARN applications

- Only acquire resource on-demand
- Configurable minimum water-level of holding resource
- Pause allocating query resource and return YARN containers when resource manager overuses YARN queue and it finds the YARN queue is busy
- Return resource when resource in-use water mark becomes lower



Even Resource Consumption among Segments

- Apache HAWQ resource manager tries its best to negotiate with global resource manager (YARN for example) to get evenly allocated resource
    - Preferred hosts are passed to global resource manager to acquire containers
    - When Apache HAWQ resource manager decides to return containers, the segments having most allocated containers are considered preferentially
- Apache HAWQ resource manager tries its best to allocate VSEGs among segments to make all available segments evenly utilized
    - Combined workload ordered index is dynamically maintained in resource pool so that the segments having lower workload and more available resource are considered in priority
    - Round-robin strategy is adopted generally to ensure that for one query execution, all possible available segments are utilized, and the segments have almost the same number of VSEGs
    - Data-locality based VSEG allocation strategy is restricted in some cases

Allocating VSEGs According to HDFS Data Locality

- Apache HAWQ holds target table's data locality information that is passed to resource manager when acquiring query resource.

- Apache HAWQ resource manager allocates VSEGs among segments if it is appropriate to follow data locality information.

    - It works for small scale queries requiring a few VSEGs only
    - If a lot of VSEGs are required for a large scale query execution, this strategy is not adopted
    - If the VSEG allocation intensifies the balance of resource usage among segments too much, this strategy is not adopted

Apache HAWQ Resource Queues

DDL for manipulating resource queues

```
CREATE RESOURCE QUEUE queue_name WITH (queue_attr_list);
ALTER RESOURCE QUEUE queue_name WITH (queue_attr_list);
DROP RESOURCE QUEUE queue_name;
```

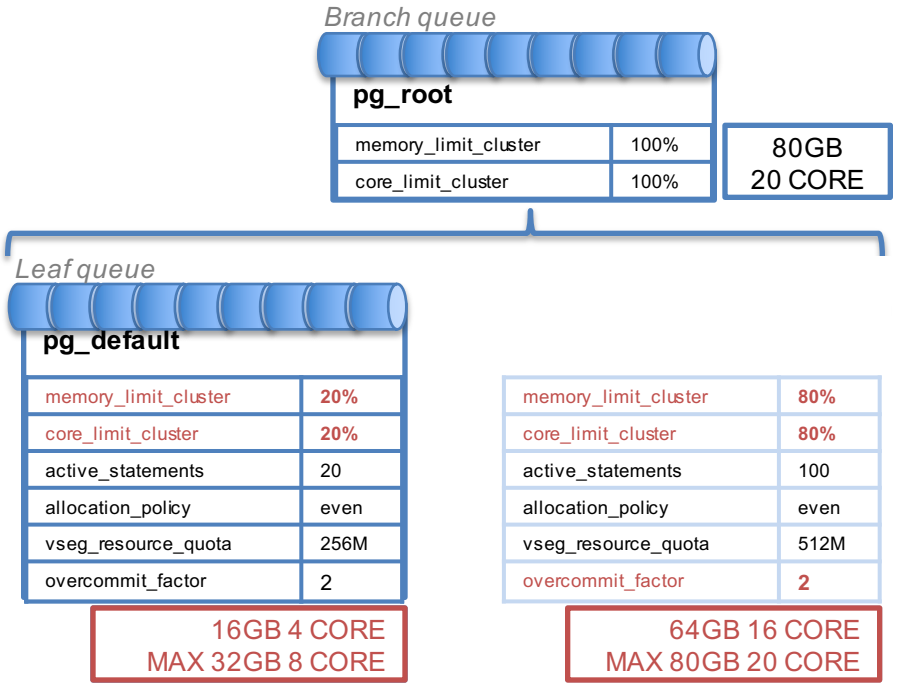
Attribute	Value
PARENT*	The parent queue's name.
ACTIVE_STATEMENTS	The limit of number of parallel active statements. Resource queue's concurrency control.
MEMORY_LIMIT_CLUSTER*	The percentage of memory resource consumable from its parent queue's capacity.
CORE_LIMIT_CLUSTER*	The percentage of virtual core resource consumable from its parent queue's capacity. *Currently, this value must be identical with* *MEMORY_LIMIT_CLUSTER*.
ALLOCATION_POLICY	The resource allocation policy, currently, only 'even' policy is supported.
VSEG_RESOURCE_QUOTA	The resource quota of one virtual segment (VSEG).
RESOURCE_OVERCOMMIT_FACTOR	The factor of overusing resource from the other resource queues.
NVSEG_UPPER_LIMIT	The absolute maximum number of VSEGs for one statement. Default value 0.
NVSEG_LOWER_LIMIT	The absolute minimum number of VSEGs for one statement. Default value 0.
NVSEG_UPPER_LIMIT_PERSEG	NVSEG_UPPER_LIMIT_PERSEG times available segment count derives the absolute maximum number of VSEGs for one statement. Default value 0.0.
NVSEG_LOWER_LIMIT_PERSEG	NVSEG_LOWER_LIMIT_PERSEG times available segment count derives the absolute minimum number of VSEGs for one statement. Default value 0.0.

** The attributes must be specified when creating a resource queue.*

Pivotal

Tree-structured Resource Queues

Resource queues' capacities are automatically estimated based on dynamic cluster resource capacity and resource queue DDL manipulations. Statements consume resource from leaf queues.




```
CREATE RESOURCE QUEUE pg_dep1 WITH(
    PARENT='pg_root',
    ACTIVE_STATEMENTS=100,
    VSEG_RESOURCE_QUOTA='mem:512mb',
    OVERCOMMIT_FACTOR=1
);
```

```
ALTER RESOURCE QUEUE pg_default WITH(
    MEMORY_LIMIT_CLUSTER=20%,
    CORE_LIMIT_CLUSTER=20%
);
```

```
CREATE ROLE demorole RESOURCE QUEUE pg_dep1;
```

```
SET ROLE demorole;

SELECT COUNT(*) FROM foo;
```

```
ALTER RESOURCE QUEUE pg_dep1 WITH(
    MEMORY_LIMIT_CLUSTER=80%,
    CORE_LIMIT_CLUSTER=80%,
    OVERCOMMIT_FACTOR=2
);
```

Pivotal

DEMO

Thank you

Pivotal