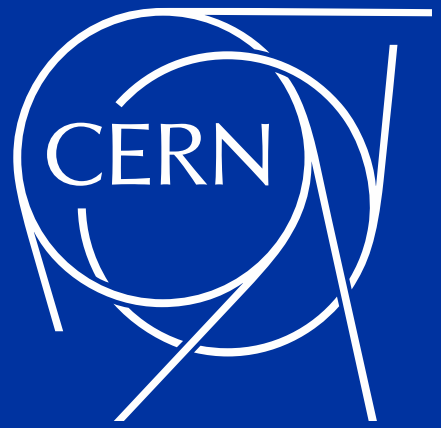


EOS Scheduler Plans

Abhishek Lekshmanan on behalf of the EOS team

Outline

- **Scheduler in EOS**
  - Introduction - GeoTree Scheduler
  - Advanced Parameters
  - Implementation
  - Performance Analysis
- **Flat Scheduler`**
  - Motivation
  - Design
  - Results
- **Conclusion and Future plans**

Current Scheduler

Introduction to GeoTree Scheduler

File Scheduler in EOS - Purpose

- Select Filesystems to place/access file replicas/stripes
  - Carried out at MGM - (GeoTreeEngine subcomponent)
  - Always chosen within placement group - Ensures a bare minimum host level failure domain
    - Groups can span across DCs
- Ensure files are evenly distributed considering
  - placement policies
  - Internal operations like draining/balancing

GeoTree Scheduler

- Ensure Geographic redundancy when groups span across DCs
- **GeoTags** for nodes corresponding to Infrastructure eg: `0513::R::0050::CQ07::a2e2917f`
  - Allows for entire subtrees to be disabled/enabled
  - For specific operations as well (placement vs drain)
- **File Placement** based on:
  - Filesystem geotag
  - State of filesystem + machine
  - Client geotag
  - File layout
  - A few admin params to control penalty
  - User supplied options for eg: placement policy(scattered, gathered), force scheduling group...

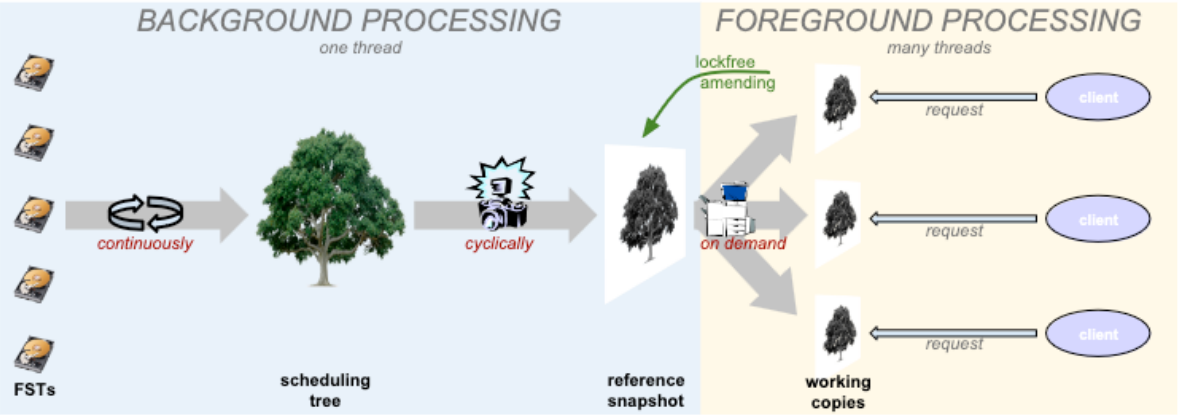
GeoTree Scheduler Parameters

- Also supports a few advanced features:
  - Proxy Scheduling (eg. Kinetic)
  - Firewall based entry point
- Fine Tuned Parameters to control
  - Upload/Download penalties for placement/access
  - Tree refresh times
  - Fill ratios...

```
[root@eospilot-ns-ip700 (mgm:master mq:master) ~]$ eos geosched show param
### GeoTreeEngine parameters :
skipSaturatedAccess = 0
skipSaturatedDrnAccess = 0
skipSaturatedBlcAccess = 0
proxyCloseToFs = 1
penaltyUpdateRate = 0
plctDlScorePenalty = 10(default) | 10(1Gbps) | 10(10Gbps) | 10(100Gbps) | 10(1000Gbps)
plctUlScorePenalty = 10(defaUlt) | 10(1Gbps) | 10(10Gbps) | 10(100Gbps) | 10(1000Gbps)
accessDlScorePenalty = 10(default) | 10(1Gbps) | 10(10Gbps) | 10(100Gbps) | 10(1000Gbps)
accessUlScorePenalty = 10(defaUlt) | 10(1Gbps) | 10(10Gbps) | 10(100Gbps) | 10(1000Gbps)
fillRatioLimit = 80
fillRatioCompTol = 100
saturationThres = 10
timeFrameDurationMs = 1500
```

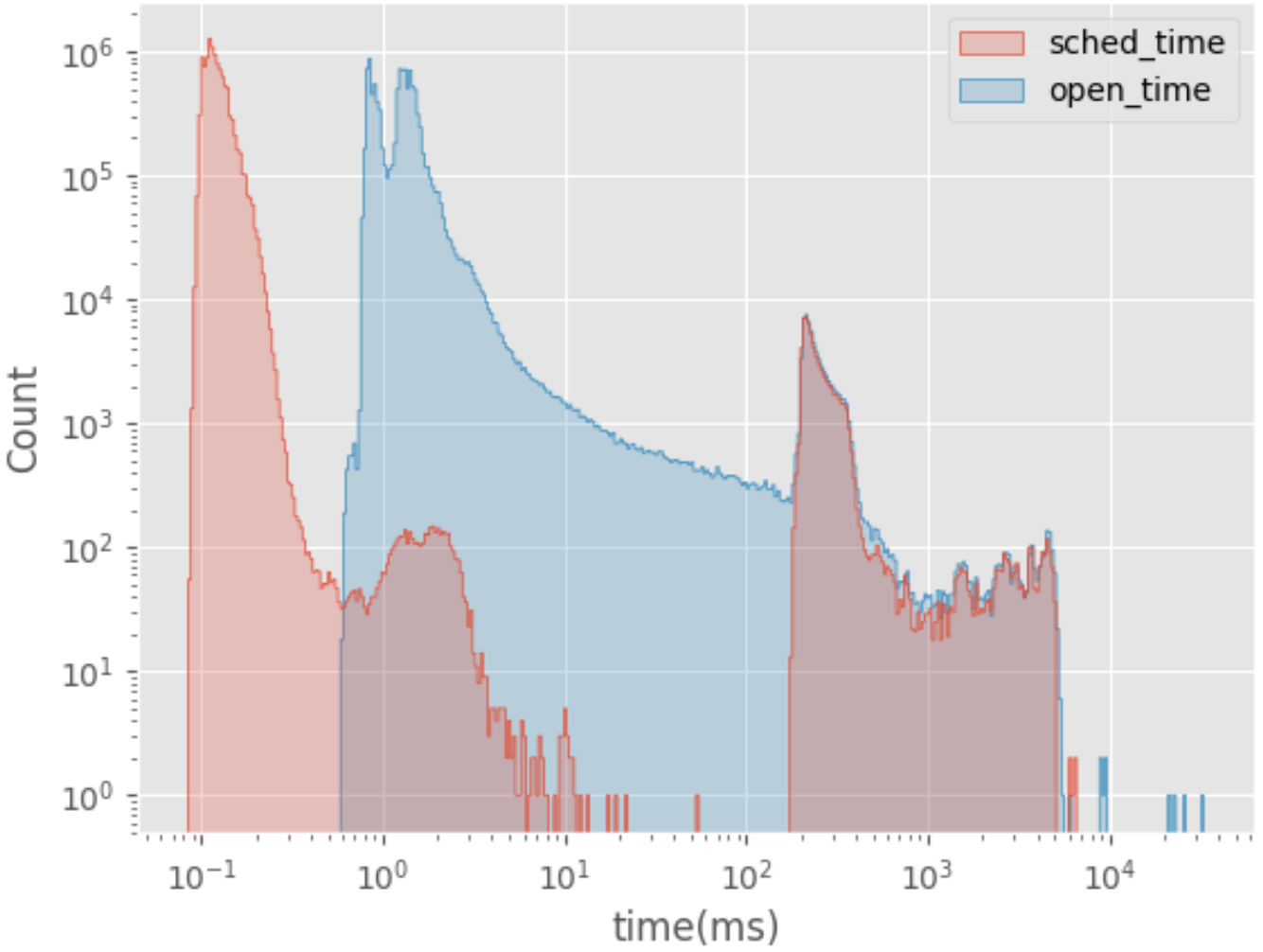
GeoTree Scheduler - Implementation

- Scheduler selects Group for placement in RR fashion (FsView Lock)
- GeoTreeEngine selects FS from within the groups
  - 2 tree structures - SlowTree having accurate representation and thread local FastTree snapshot
  - Double Buffer Mutex pattern to update TreeInfo



Performance Analysis

- Open times are logged with a per subroutine time at INFO level
- Analyzing MGM Write **Open times** in an EOS physics instance over a 10-day period (11.1 million events)
  - Avg Open Write time: **3.96ms** (stddev- 58.76 Median 1.27)
  - Avg FilePlacement time: **2.34 ms** (stddev - 52.17 Median .112)
  - 78K events have opens > 100ms
  - **87%** of 100ms+ open writes caused by FilePlacement times



Flat Scheduler

New Scheduler Motivation

- Have **Simpler** Scheduling Strategies
  - No need to pay for the cost of multiple site geo scheduling when dealing with smaller sites
  - Have **configurable scheduling algorithms**
  - Understand heterogenous disk sizes and allow for **weight based scheduling**
- Ability to fallback to classic GeoScheduler or if it is requested
- Loosely couple with other dependencies so that it is easy to unit test and benchmark
  - Dropping the dependency on **MQ**
- **Performance** - Can we perform better with these reduced constraints?

FlatScheduler Data Types

- Simple Datatypes:
  - **Disks:** ID corresponding to FSID, atomic status and weights
  - **Buckets:** Any other element in Storage Hierarchy
    - **Root,** Site, Room, Rack, **Group...**
    - Negative ID
    - Contains a list of items which may be buckets or disks
    - Total weight is the weight of elements underneath
  - **ClusterData:** List of Buckets and Disks

- **RuleMap**
  - An array of rules of how many replicas to be chosen at each level, -1 denotes take as many items as replicas requested
  - Easy to build frontends that can build this rule map

Cluster Manager

- Essentially holds and constructs the **ClusterData**
- A simple builder to build any hierarchy and create this cluster data, delegated to a builder class - 2 methods
  - adds a **Bucket** of a given type (room, group etc.) to its Parent
  - a **Disk** with Disk Params (default weight is size in TiB), parent Bucket ID
- Have a flat FsView based builder already implemented, that'll build the current EOS view of storage in a flat fashion
  - Holds a list of **ClusterMgrs**
  - Flat Hierarchy - Root element containing a list of Group IDs
  - Lock Free for readers wrt Cluster Map updates (using Atomic Ptrs and deferred reclamation)
- Easy to extend to geotag based or a more hierarchical aware builder in the future

FlatScheduler

- Simple interface
- Holds a list of **PlacementStrategies**
  - Placement Strategies decide how to choose the next elements (buckets, disks)
  - Passed as argument, so ability to decide on a per op basis
- Scheduling basically act of choosing n replicas at each bucket type- BFS like walking through each bucket picking replicas
- Default rule map is just to pick single group from ROOT and find disks

```
class FlatScheduler {
public:

  FlatScheduler(size_t max_buckets);
  FlatScheduler(PlacementStrategyT strategy, size_t max_buckets);

  PlacementResult schedule(const ClusterData& cluster_data,
                           PlacementArguments args);

private:
  PlacementResult scheduleDefault(const ClusterData& cluster_data,
                                  PlacementArguments args);

  std::array<std::unique_ptr<PlacementStrategy>, TOTAL_PLACEMENT_STRATEGIES>
      mPlacementStrategy;
  PlacementStrategyT mDefaultStrategy{PlacementStrategyT::Count};
};
} // namespace eos::mgm::placement
```

Current Results

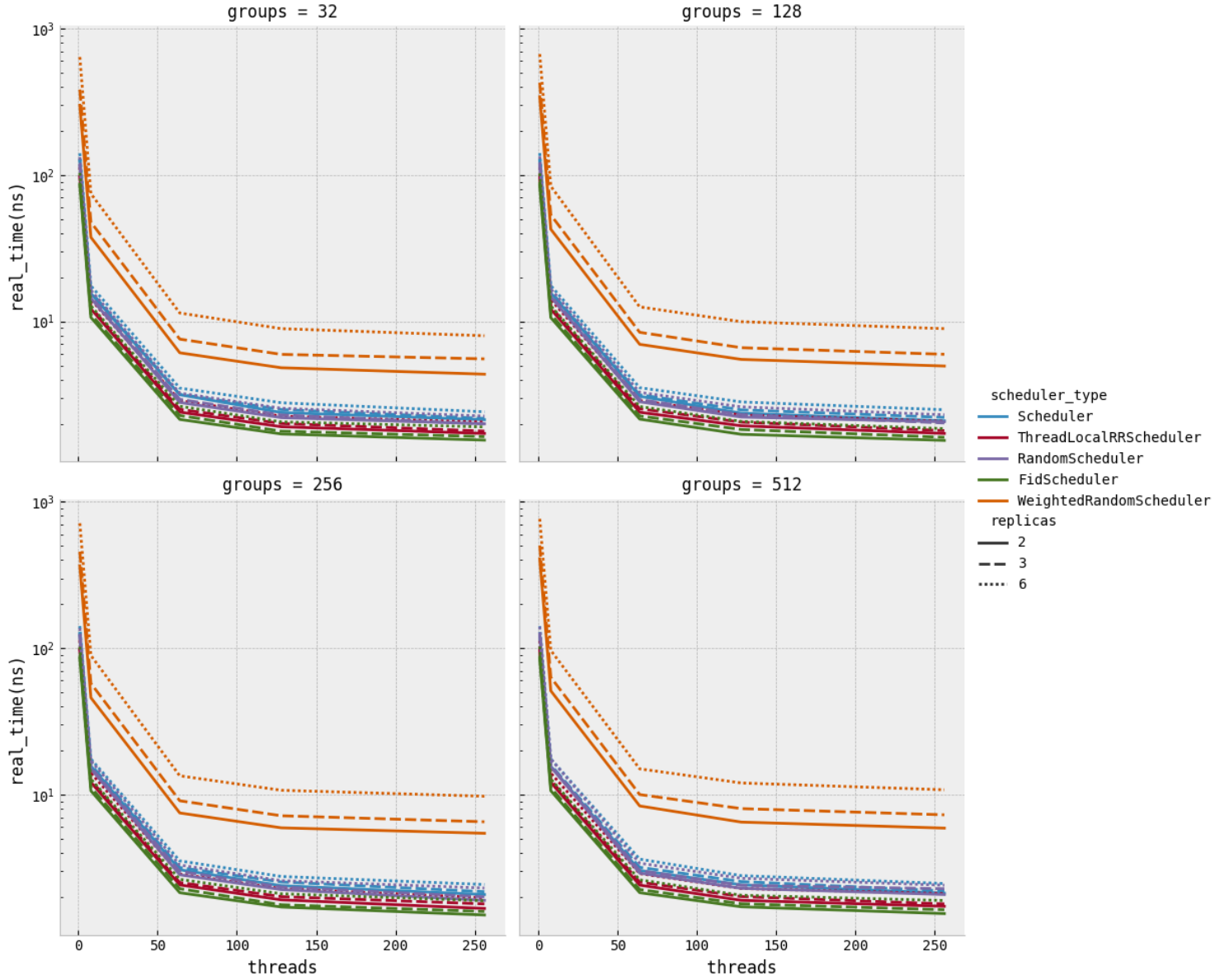
```
[root@abhi-dev-cc7 ~]# eos group ls
```

type	name	status	N(fs)	dev(filled)	avg(filled)	sig(filled)	balancing	bal-shd
groupview	default.0	on	7	0.00	48.02	0.00	idle	0
groupview	default.1	on	4	0.00	48.02	0.00	idle	0
groupview	default.2	on	3	0.00	48.02	0.00	idle	0

```
[root@abhi-dev-cc7 ~]# eos ls -l /eos/rw/wrtests/wtscheduler
[root@abhi-dev-cc7 ~]# eos ls -l /eos/rw/wrtests/rrscheduler
[root@abhi-dev-cc7 ~]# for i in {1..100}; do eos cp -s 1B /eos/rw/wrtests/wtscheduler/f$i?eos.schedulingstrategy=weightedrandom; done
[root@abhi-dev-cc7 ~]# for i in {1..100}; do eos cp -s 1B /eos/rw/wrtests/rrscheduler/f$i?eos.schedulingstrategy=rr; done
[root@abhi-dev-cc7 ~]# for i in {1..100}; do eos -j fileinfo /eos/rw/wrtests/rrscheduler/f$i | jq -r ".locations[].schedgroup" >> schedgrouprr.txt; done
[root@abhi-dev-cc7 ~]# for i in {1..100}; do eos -j fileinfo /eos/rw/wrtests/wtscheduler/f$i | jq -r ".locations[].schedgroup" >> schedgroupwt.txt; done
[root@abhi-dev-cc7 ~]# cat schedgrouprr.txt | sort | uniq -c
     33 default.0
     33 default.1
     34 default.2
[root@abhi-dev-cc7 ~]# cat schedgroupwt.txt | sort | uniq -c
     48 default.0
     29 default.1
     23 default.2
```

Benchmark Results

- Benchmarks were done with
  - AMD EPYC 7302 16 core (64 threads)
  - varying number of groups (32->512)
  - Replica count (2,3, raid6)
  - Different scheduling strategies for the Flat Scheduler ie. RoundRobin, Weighted Random
  - Google Benchmark library which runs benchmarks in a tight loop
- As the thread count kept increasing we see increasing scalability
- Nearly lock-free code path, FsView filled in flat lists which are lock free (RCU like algorithm)
- Single Thread performance still ~1us

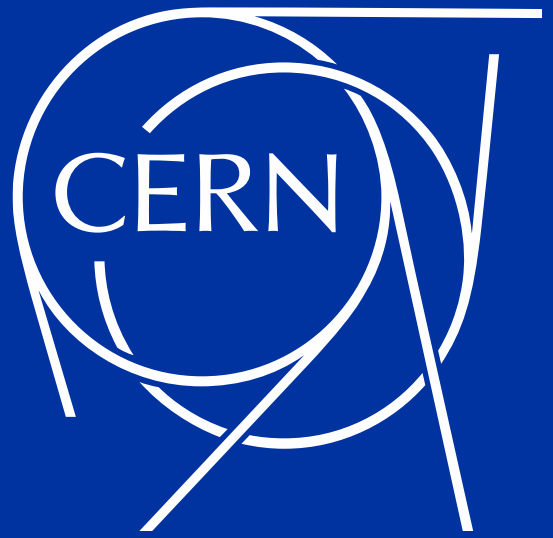


Conclusions & Future Work

- A simple flat scheduler with a few placement strategies will make it soon to EOS MGM
  - Policies to be configured on a space/directory/op level
- Most of the subcomponents & strategies can be built and unit tested in isolation
  - Makes debugging and functionality testing easier
  - Mock Fixtures have already been written to test functionalities
- Current Benchmarks show promising results
  - Evaluate how these translate in real workloads in test instances
-

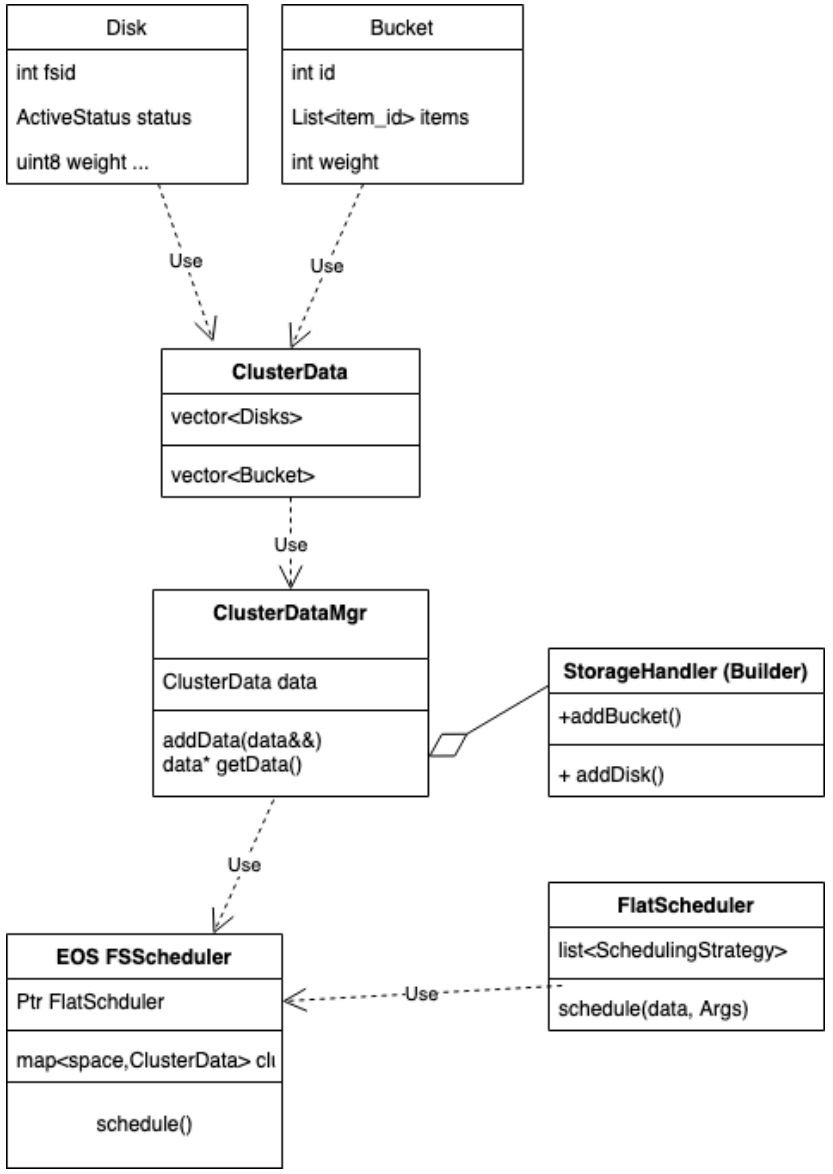
Future Work

- A static hierarchy based rules could be added in the Future
  - Interface to be defined, but easy to build a mini crush map like language
  - Not heavy on state, potential for passing down scheduling decisions to other gateway nodes (& even clients?)
- Some advanced GeoScheduler features that is not planned in the immediate timeline - ProxyGroups
  - Firewall EntryPoint Scheduling
  - Parameters like Plct/Access Ul/Dl Penalties
  - Interested in community feedback on the usability of these!


CERN

Class Diagram

- **Disks** and **Buckets** the main data types
- ClusterData just holds a vector of these data
- ClusterDataMgr actually manages the data
  - Epochs every time the structure of data changes
  - Lock Free for readers, uses Atomic Unique Ptrs and deferred GC strategy
- FlatScheduler the main class that does all the logic
  - Scheduling delegated to strategy classes which ultimately decide how the data is selected



Concurrency Interlude - Publishing Pointer

- **Pointer** loads and stores are atomic (x86)
  - However nothing explicit about the instruction reordering
  - Compilers and hardware allowed to freely reorder instructions
- Introducing the concept of an Atomic Unique Ptr
  - Construction not thread safe, atomic loads
  - When resetting the pointer, we don’t remove the old pointer, instead it is returned and the caller has to hold on to this and find a sufficiently safe point to GC
- Performance equivalent to a regular unique pointer in comparison to a Atomic SharedPointer

Benchmark results - AtomicUniquePtr

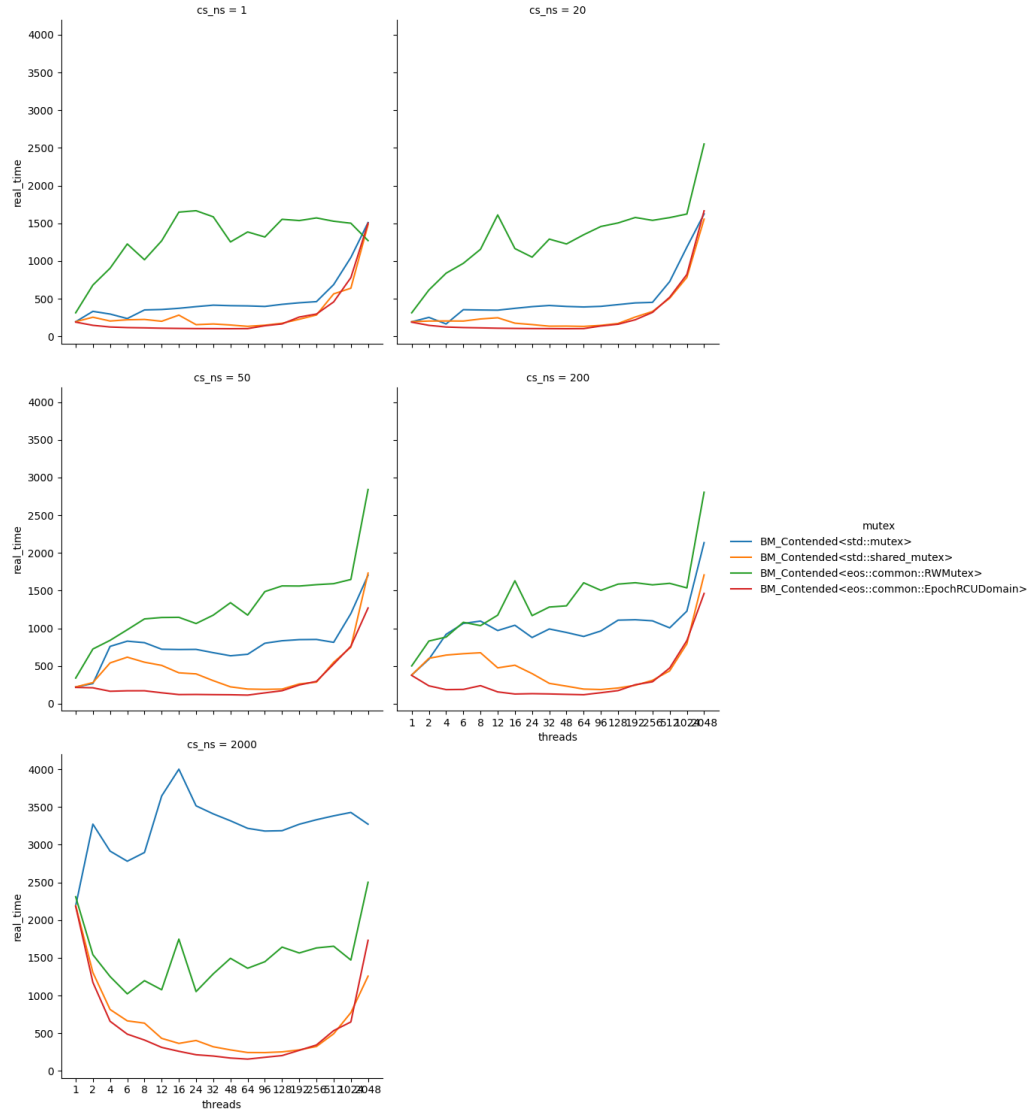
```
Running ./test/microbenchmarks/eos-atomic-ptr-microbenchmark
Run on (64 X 1798.87 MHz CPU s)
CPU Caches:
  L1 Data 32 KiB (x32)
  L1 Instruction 32 KiB (x32)
  L2 Unified 512 KiB (x32)
  L3 Unified 16384 KiB (x16)
Load Average: 0.14, 0.08, 0.01
-------------------------------------------------------------------------------------------
Benchmark                                    Time             CPU   Iterations UserCounters...
-------------------------------------------------------------------------------------------
BM_AtomicUniquePtrGet/real_time/threads:1        0.306 ns        0.306 ns   1000000000 frequency=3.26319G/s
BM_AtomicUniquePtrGet/real_time/threads:256      0.007 ns        0.599 ns   135805664256 frequency=145.081G/s
BM_UniquePtrGet/real_time/threads:1              0.308 ns        0.308 ns   1000000000 frequency=3.24369G/s
BM_UniquePtrGet/real_time/threads:256            0.007 ns        0.600 ns   156309076224 frequency=145.156G/s
BM_SharedPtrCopy/real_time/threads:1              10.7 ns         10.6 ns     64106208 frequency=93.8895M/s
BM_SharedPtrCopy/real_time/threads:256           0.111 ns         9.43 ns   6726456832 frequency=9.03931G/s
BM_AtomicSharedPtrGet/real_time/threads:1         25.8 ns         25.7 ns     26865129 frequency=38.8072M/s
BM_AtomicSharedPtrGet/real_time/threads:256       45.1 ns         3274 ns     22903808 frequency=22.149M/s
```

So we have a new pointer - how do we free?

- For deferred memory reclamation there are some common techniques
  - RCU - Read Copy Update - a non blocking Reader pattern which kernel uses a lot as a synchronization primitive
  - Hazard Pointers
- All of these techniques essentially track readers in a way that we know a safe point to reclaim the memory

Mutex Benchmarks

- Mutex benchmarks with varying critical sections



Implementation

- At the lowermost layer we've a **Disk** that has a integer fsid; atomic config/active status, weight and filled%
- The rest of the storage hierarchy can be represented by a **Bucket**; here we use a negative id; a bucket contains a list of ids representing things below its hierarchy
- ClusterData this way essentially contains a flat list of buckets and disks
  - An Epoch version tracks the version of Clusterdata
  - Any changes in the shape of the Cluster itself, ie. Addition/move of nodes/disks will need a new version of Clusterdata
  - Regular activities like disks being offline etc do not need a change of cluster data itself, so requires no scheduler pauses