Slurm User Group – September 2024

Matthieu Hautreux, Larry Pezzaglia

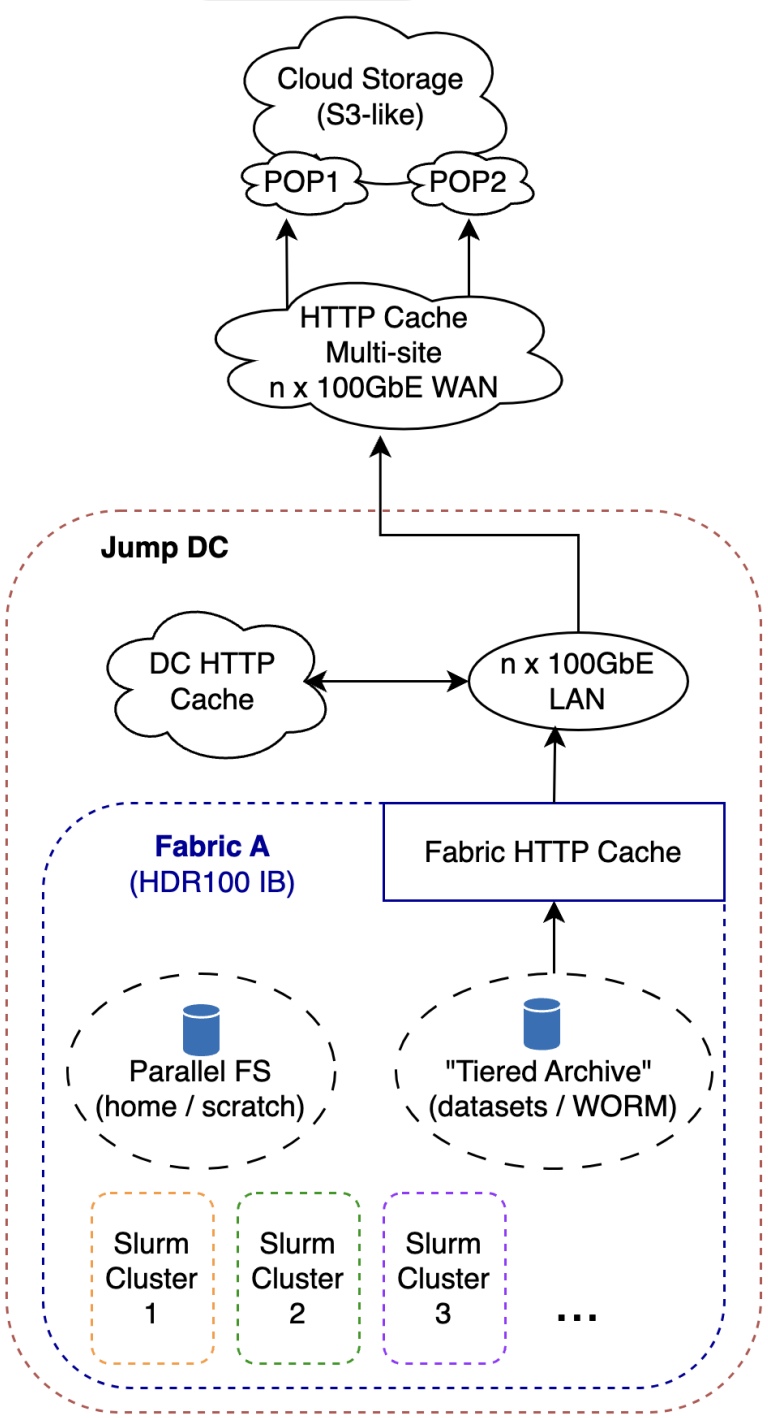
Jump Trading

- Jump Trading is a proprietary trading firm committed to world-class research
- Full migration to Slurm in 2023
- Enabling research at scale requires automation, flexibility, self-service systems, and a mindset of constant improvement

Slurm at Jump Trading

Agenda

- **Jump Trading HPC overview**
- Workload Characteristics and Slurm migration
- Slurm features sponsored by Jump
- Slurm enhancements for high job throughput
- What's next ?

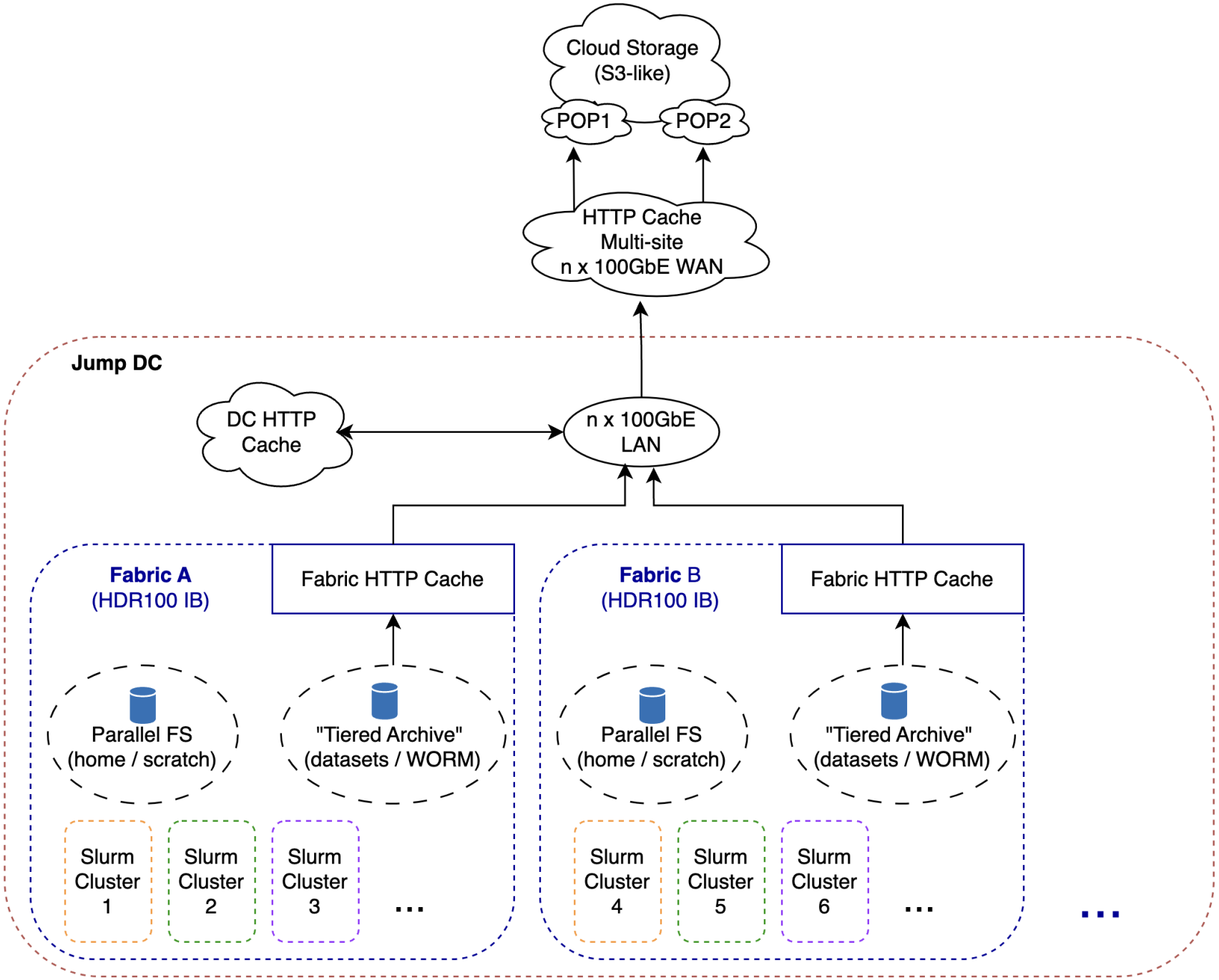


Jump HPC Fabric

- Textbook HPC components
  - RDMA-capable fabric
  - Parallel filesystem
  - Workload manager (Slurm)
- Add: Many Slurm clusters (1 or 30+ per fabric)
  - Separate clusters per internal team or project
  - Dynamically resizable by users via self-service automation
- Add: “Tiered Archive” storage system[1]
  - Read-only filesystem presentation (CVMFS)
  - Backed by HTTP caches and cloud storage
  - rsync-like write interface for users

[1] https://indico.cern.ch/event/1377701/contributions/5863778/attachments/2837970/4959865/HEPIX-CVMFS-Presentation-JUMP.pdf

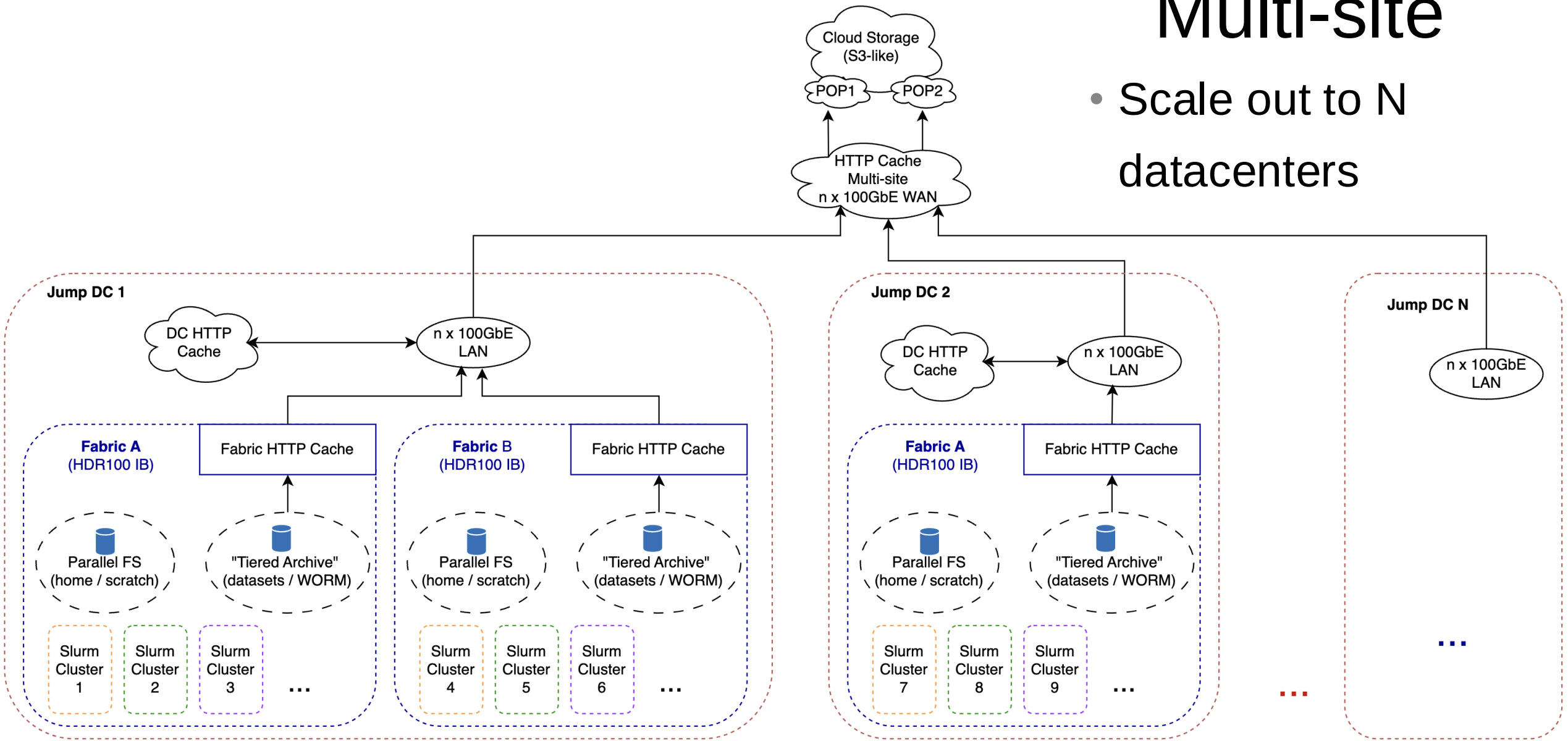
Multi-fabric



- Scale out to N fabrics in a DC
- Contains blast radius of fabric and parallel FS issues
- Data sharing among fabrics via Tiered Archive only

Multi-site

- Scale out to N datacenters



Slurm at Jump Trading

Agenda

- Jump Trading HPC overview
- **Workload Characteristics and Slurm migration**
- Slurm features sponsored by Jump
- Slurm enhancements for high job throughput
- What's next ?

Workload Characteristics

- **Most** jobs are short, **single-core**, and data-intensive
  - Nodes run many tasks from many different jobs and users
  - Expectation of “immediate” task starts when resources are available
  - High Slurm throughput required to keep clusters full
- **Most** pipelines orchestrated by in-house workflow generator
  - Complex workflows, thousands of work units, submitted in job arrays
  - Abstracts away Slurm details
  - Performance sensitive to delta between submit time and start time

Workload Characteristics

Many exceptions and complications

- Complicated resource requests (both node-local and cluster-wide), including GPUs (both partial and exclusive access)
- Large jobs (including multi-node) versus small, short jobs
  - Some large jobs use MPI
  - Want to avoid starvation of large jobs by small jobs
- Deadline-sensitive pipelines (e.g., must finish by midnight)
- Some “power users” prefer direct Slurm access

Slurm Migration Playbook

- Provide teams with a Slurm cluster alongside the incumbent batch system's cluster
- Teams grow (via self-service automation) their Slurm cluster and shrink the incumbent cluster, eventually to zero
  - Bump in the road? Reverse this process
  - Many issues triggered by high throughput and only surfaced at scale
- Migration completed by end of 2023

Slurm at Jump Trading

Agenda

- Jump Trading HPC Overview
- Workload Characteristics and Slurm Migration
- **Slurm features sponsored by Jump**
- Slurm enhancements for high job throughput
- What's next ?

Sponsored Features

Challenge	Slurm Solution
Many dynamic logical clusters, user-sized	**Dynamic nodes (22.05) – Sponsored by Jump** • No node definitions in slurm.conf • Pairs well with configless mode
Existing user practice: “soft” feature requests	**Preferred Features (22.05) – Sponsored by Jump** • --prefer: Like –constraint, but best effort • Users tag nodes with features (Jump automation) • Enables “spillover” (reserve N nodes for a pipeline, but allow use of other free nodes)
Per-task resource requirements • Many GRES (including GPU) requests previously were per-node in Slurm • But our nodes are heterogeneous and can run tasks from many jobs	**--tres-per-task (23.02) – Sponsored by Jump** • Per-task requests like resources (GPUs, etc.)

Sponsored Features

Challenge	Slurm Solution
Existing user practice: time-slicing GPUs	**GPUs sharding (22.05) – Sponsored by Jump** • Users may request full GPUs or shards of GPUs • Some teams use MIG (Multi-Instance GPU) instead
Granular control over GPU shards	**Per-task shards (23.11) – Sponsored by Jump** • MULTIPLE_SHARING_GRES_PJ SelectType • Jobs need multiple “sharing” (e.g., shards) GRES per node **Shard {anti-,}affinity (23.11) – Sponsored by Jump** • Require shards on same/different physical GPUs • Toggleable per job

Sponsored Features

Challenge	Slurm Solution

Runtimes hard to estimate
• Research problem maps well to one job array task per data slice
  • Data slices are different sizes
  • External factors (e.g., FS performance)
• Users react by overestimating --time by 100x
	Soft Time Limits (23.11) – Sponsored by Jump
	• **--time-min** used for backfill
	• **--time** still used for hard time limit

High job throughput • Core counts per node keep increasing • Need other “expensive” features (preemption, backfilling) enabled • Avoid starvation of large jobs	Development in progress – **Sponsored by Jump**

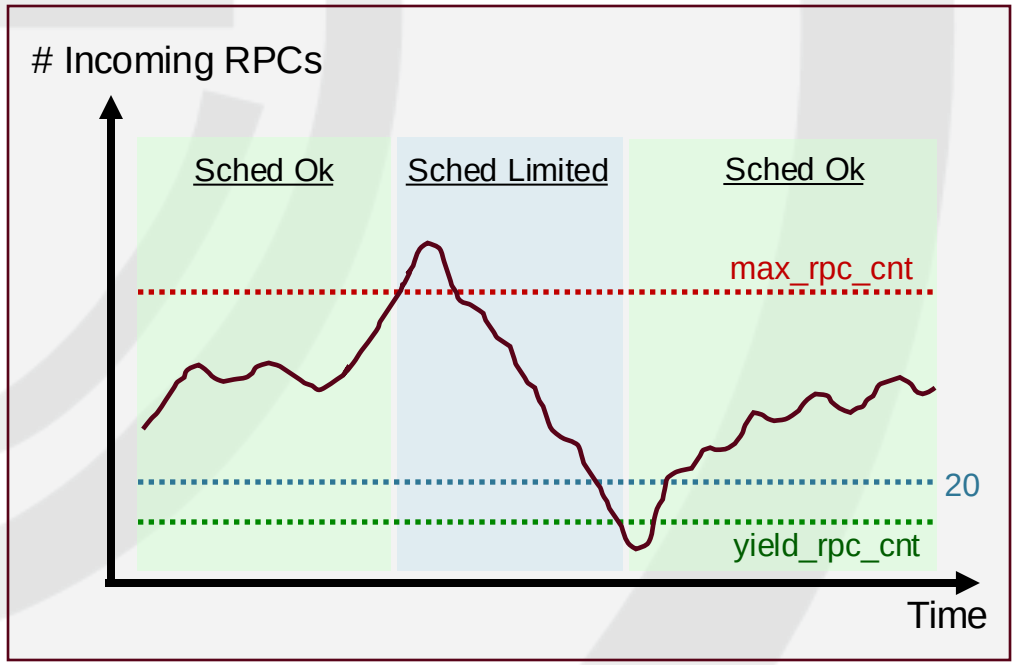
Slurm at Jump Trading

Agenda

- Jump Trading HPC Overview
- Workload Characteristics and Slurm Migration
- Slurm features sponsored by Jump
- **Slurm enhancements for high job throughput**
- What's next ?

Slurm enhancements for High-throughput

- Slurm controller locks contention decrease High-throughput perf
  - Slurm Controller is highly/multi-threaded
  - Slurm internal data structures are protected by read/write locks
    - Internal Slurm components acquire the read / write locks they need
    - Internal Slurm components timing may vary depending on the scale / complexity
  - Locks arbitration logic mostly based on the # of RPC
    - Schedulers (Main or backfill) stop / yield under high incoming RPC load until reaching a low level again
      - **yield_rpc_cnt = MAX((max_rpc_cnt / 10), 20)**
    - Heavy RPC processing serialized+delayed under outgoing RPC load
  - This may lead to suboptimal scheduling perf under HT
    - Free/available resources not being used fast enough



Slurm enhancements for High-throughput

- Hidden existing **enable_rpc_queue** parameter
  - Create a dedicated processing queue+thread for certain types of RPCs
  - Experimental / incomplete native logic in Slurm
  - Queued RPCs not counted in **max_rpc_count**

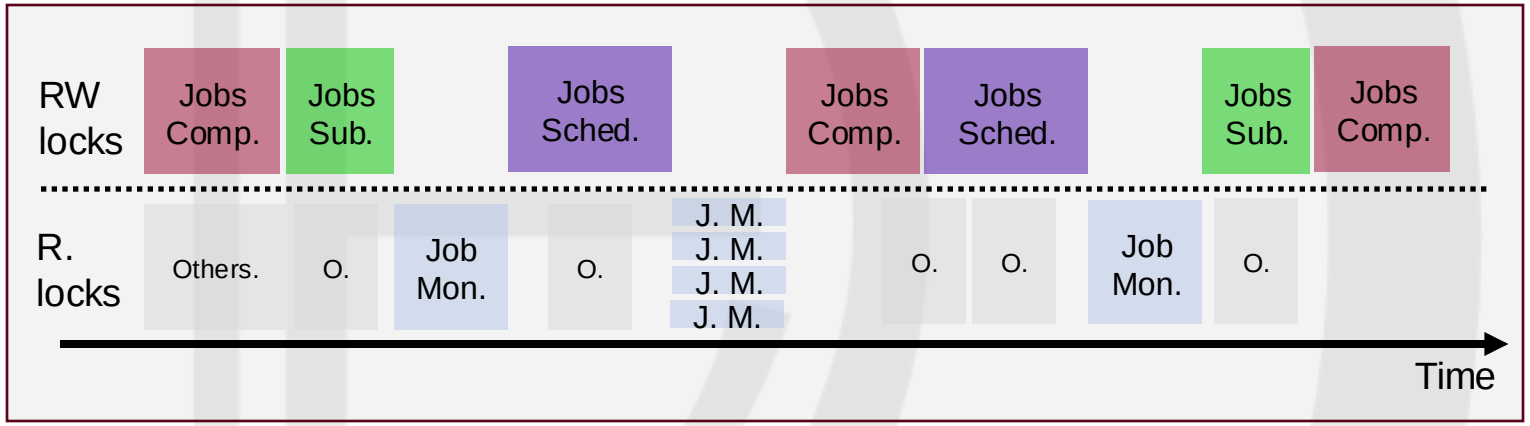
- Local enhancement of the rpc_queue logic
  - Additional RPC types managed via rcp_queue
  - **per-RPC-type credits|time-based concurrency** model
  - Associated tunables per RPC type
    - max_per_cycle, max_usec_per_cycle, yield_sleep, interval, max_queued, …
  - per-RCP-type disabling is possible
    - (read-lock only RPCs may be better served with the native logic)

Queue-enabled RPCs :

REQUEST_JOB_INFO
REQUEST_JOB_USER_INFO
REQUEST_JOB_INFO_SINGLE
REQUEST_FED_INFO
REQUEST_NODE_INFO
REQUEST_PARTITION_INFO
REQUEST_COMPLETE_PROLOG
REQUEST_COMPLETE_BATCH_SCRIPT
REQUEST_JOB_STEP_CREATE
MESSAGE_NODE_REGISTRATION_STATUS
REQUEST_SUBMIT_BATCH_JOB
REQUEST_STEP_COMPLETE

Slurm enhancements for High-throughput

- Enhanced rpc_queue logic is not perfect but greatly helps
  - More deterministically spread locks time across the main areas of activities
    - Jobs submissions
    - Jobs states monitoring
    - Jobs scheduling
    - Jobs completion & epilog



  - Queued RPCs still keep their TCP socket opened
    - Need to properly configure queues tunable, especially max queue size (to trigger client backoff retries and/or abort)

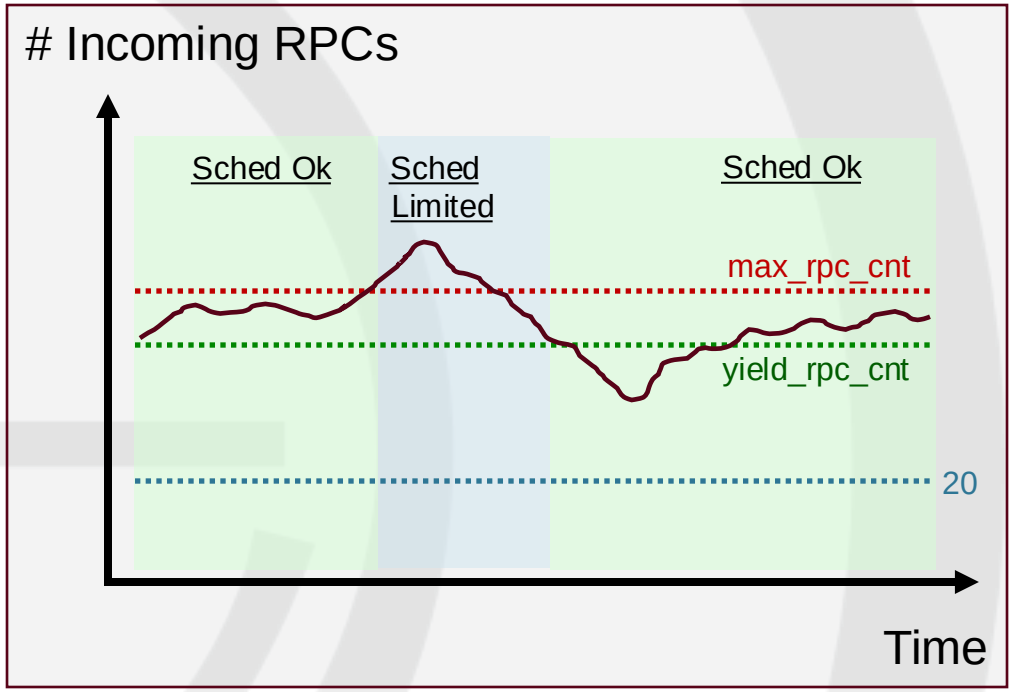
- # of RPCs still used in parallel
  - To help throttle down sched during incoming RPC peaks
    - For non-rpc_queue aware RPCs
    - For rpc_queue enabled RPCs while pushing them to their queue

slurmctld - enqueued RPCs - max processed count per cycle - by message type

message_type	average_microsec_per_rpc	max_per_cycle ↓
REQUEST_JOB_INFO_SINGLE	39	4096
REQUEST_FED_INFO	14	3553
MESSAGE_EPILOG_COMPLETE	19	2493
REQUEST_COMPLETE_BATCH_SCRIPT	151	2284

Slurm enhancements for High-throughput

- Enhancement of the sched yield / stop thresholds conf
  - Keep sched enabled under heavy yet manageable RPC load
  - Make yield_rpc_cnt configurable, example :
    - **max_rpc_cnt = 150**
    - **yield_rpc_cnt = 100**
      - (greater than the default max of 20)



- Other modifications have been made to help improving sched performances like
  - Reducing the amount of time required per backfill job scheduling attempt taking some shortcuts
    - May not apply to all workloads
  - Reusing free resources as soon as possible even when nodes are completing
    - to better handle large SMP nodes with single-core jobs

Slurm at Jump Trading

Agenda

- Jump Trading HPC Overview
- Workload Characteristics and Slurm Migration
- Slurm features sponsored by Jump
- Slurm enhancements for high job throughput
- **What's next ?**

What's next ?

- SchedMD has integrated Jump **rpc_queue** enhancements in **24.05**
  - Still considered experimental and hidden
  - A new rpc_queue.yaml file enables to configure the various per RPC type tunings
  - Other improvements added and/or ongoing by SchedMD to further improve HT in **24.05** and **24.11**
- SchedMD has reviewed some of our scheduling modifications
  - And is actively working on alternatives to further improve performances in **24.11**

    (part of the in-progress development sponsored by Jump)
- Additional local patches / features (not discussed in this presentation) maintenance
  - Let's discuss later if some of the following subjects matter to you
    - Automatic association of a default Account to jobs belonging to undefined users
    - Direct step-cancelation in OOM situations, cgroup(v1) time-sharing issues between jobs
    - Opt-in memory overcommitment in jobs to use in best-effort more memory than allocated when possible

Slurm at Jump Trading

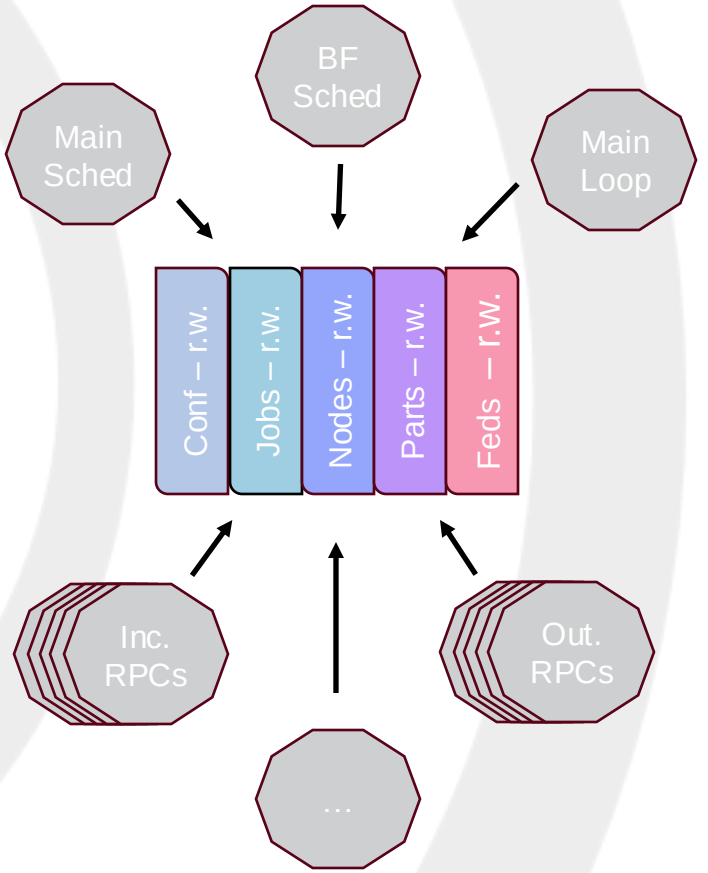
Q&A

Slurm at Jump Trading

Backup Slides

Slurm enhancements for High-throughput

- Context
  - Slurm Controller is highly/multi-threaded
    - Incoming RPCs threads
    - Outgoing RPCs threads (agents for compute nodes actions requests)
    - Backfill scheduler thread
    - Main sched agent thread
    - Main loop/house-keeping thread
    - …
  - Slurm internal data structures are protected by read/write locks
    - Configuration / Jobs / Nodes / Partitions / Federations
    - Locks ensure that states stay consistent across the variety of handled events
  - Slurm threads acquire the read / write locks they need when needed
    - Potentially preventing other concurrent components to do their job
    - Potentially being delayed / starved by other components



Slurm enhancements for High-throughput

- Context
  - Internal Slurm components acquire the read / write locks they need
    - Potentially preventing other concurrent components to do their job
    - Potentially being delayed / starved by other components
  - Internal Slurm components timing may vary depending on
    - The size and complexity of the cluster
      - amounts of resources / generic resources / GPUs / licenses / …
    - The number of active/recent jobs
      - in different states including running, pending, recently terminated
    - The specificities of configuration
      - # of partitions, # of various nodes weights, preemption / oversubscription options, …

Slurm enhancements for High-throughput

- Context
  - Some potential large locks-time consumers are also natively throttled
    - One-by-one serialization in various RPCs processing
      - REQ_RESOURCE_ALLOCATION, REQ_SUBMIT_BATCH_JOB, REQ_KILL_JOB, …
      - _throttle_start / _throttle_fini with extra usleep to help locks rotation between RPCs
        - Bigger usleep when outgoing RPCs/Agents is high (***LOTS_OF_AGENTS***)
    - Tend to increase the # of associated incoming RPC threads in case of burst
    - Thus, triggering the yield/stop of the schedulers
  - Other heavy locks-time consumers appear at some scales
    - RPC_JOB_INFO / RPC_JOB_USER_INFO with (dozens of) thousands of jobs, …

Slurm enhancements for High-throughput

- Slurm in high-throughput mode is pushing the model to its limits
  - High submission and/or completion rates means lot of incoming RPCs
  - Schedulers yielding almost all the time
  - Not enough locks-time to schedule enough jobs
  - Global usage of resources lower than expected / possible
  - Schedulers even with enough cycles, have difficulties to keep the resources fully used under high turn-over pressure

Slurm enhancements for High-throughput

- **rpc_queue** enhancements at Jump
  - New logic+tunings to add a **credits|time-based concurrency** model
    - For each RPC type, the following tunings can be defined :
    - **max_per_cycle** : # of RPCs that can be processed in a single batch
    - **max_usec_per_cycle** : max amount of microsecs for a single batch
    - **yield_sleep** : amount of microsecs to yield between 2 batches under pressure
    - **interval** : amount of microsecs to sleep between 2 batches with no pressure
    - **max_queued** : max # of pending RPCs in the queue before backpressure
    - **hard_drop** : indicates if backpressure should trigger retries on client side or not
  - Allow **additional RPCs** to be used with **rpc_queue**
    - To cover observed major elements (exp: epilog completion)

Slurm enhancements for High-throughput

- Example of sched modification for HT on large SMP nodes
  - Slurm Schedulers avoid scheduling new jobs on completing nodes
  - Large SMP nodes tend to always be completing under HT situations
  - Large SMP nodes are only eligible for jobs when not a single job is currently ending / completing
  - This can reduce global efficiency, as free cores/mem can not be used
  - **bf_ignore_cg_state** to allow the backfill scheduler to schedule jobs on completing nodes