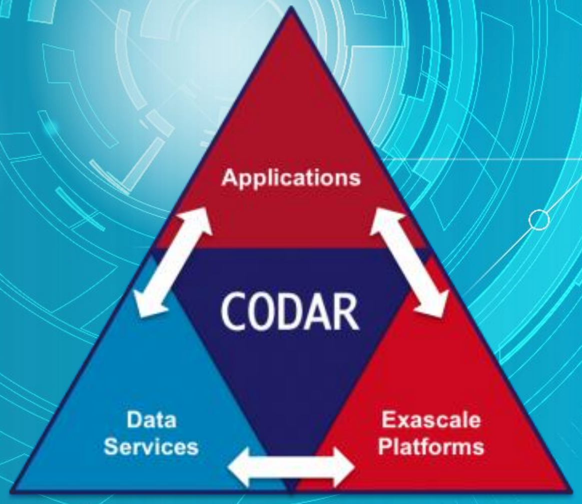
Chimbuko: a workflow-level performance anomaly detection system for HPC

Approved for public release



EXASCALE COMPUTING PROJECT

Christopher Kelly

Computational Science Initiative

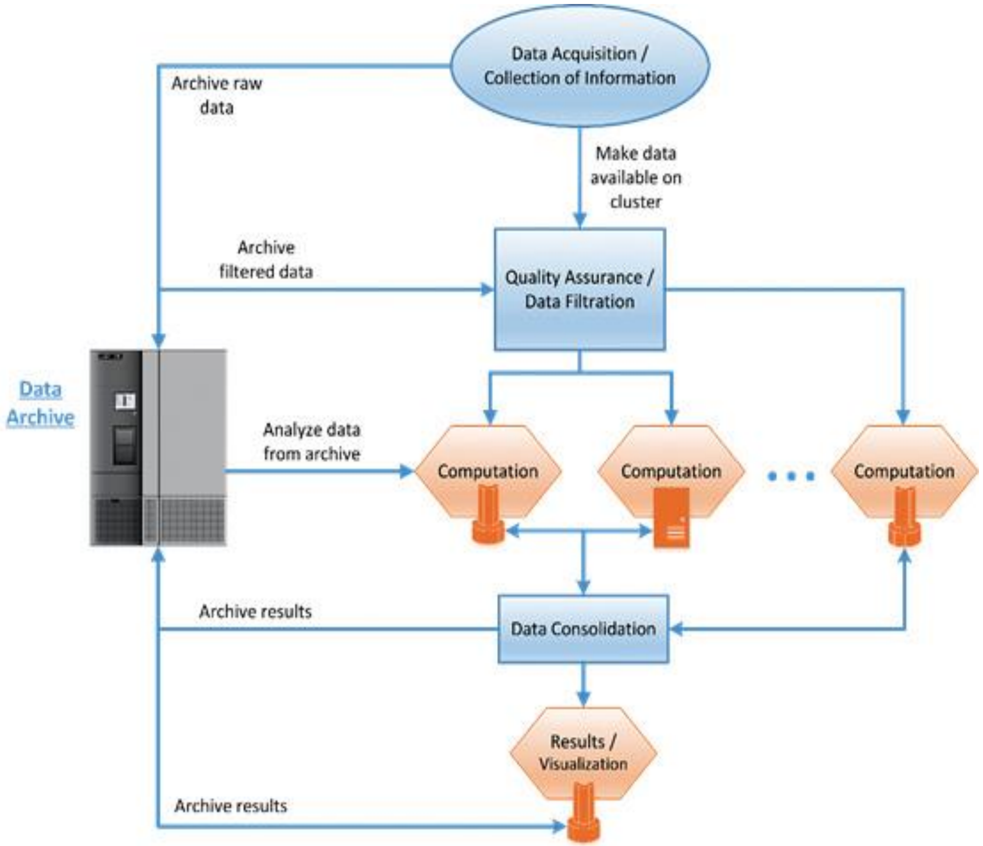
Brookhaven National Laboratory

Brookhaven National Laboratory

U.S. DEPARTMENT OF ENERGY | Office of Science

Introduction

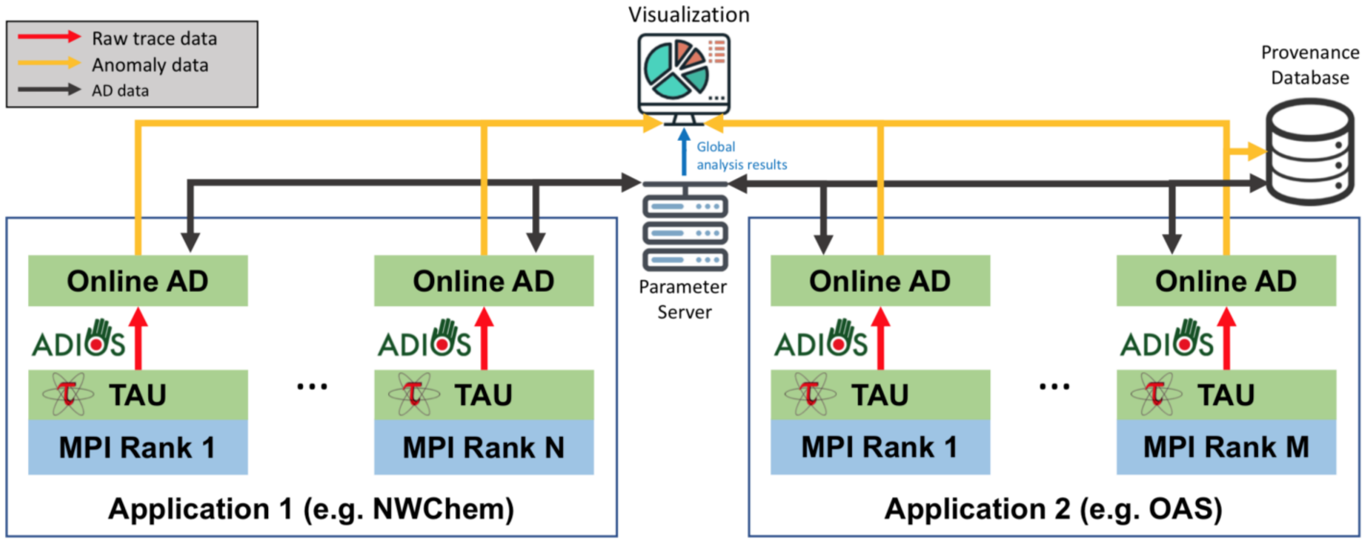
- Modern HPC workflows typically comprise multiple coupled elements running simultaneously.
- Understanding the behavior of a complex workflow running at-scale on a supercomputer is very challenging
  - Built-in timing/profiling can highlight areas of potential optimization but cannot identify root cause.
  - Capturing detailed trace data for root-cause analysis can only be done at small scale as data volumes quickly become overwhelming.
  - Small-scale analysis may not capture "stochastic" effects appearing only at scale, such as resource contention between workflow elements or hardware-driven anomalies.



EXASCALE COMPUTING PROJECT

Chimbuko : "*Place of Origin*" (Swahili)

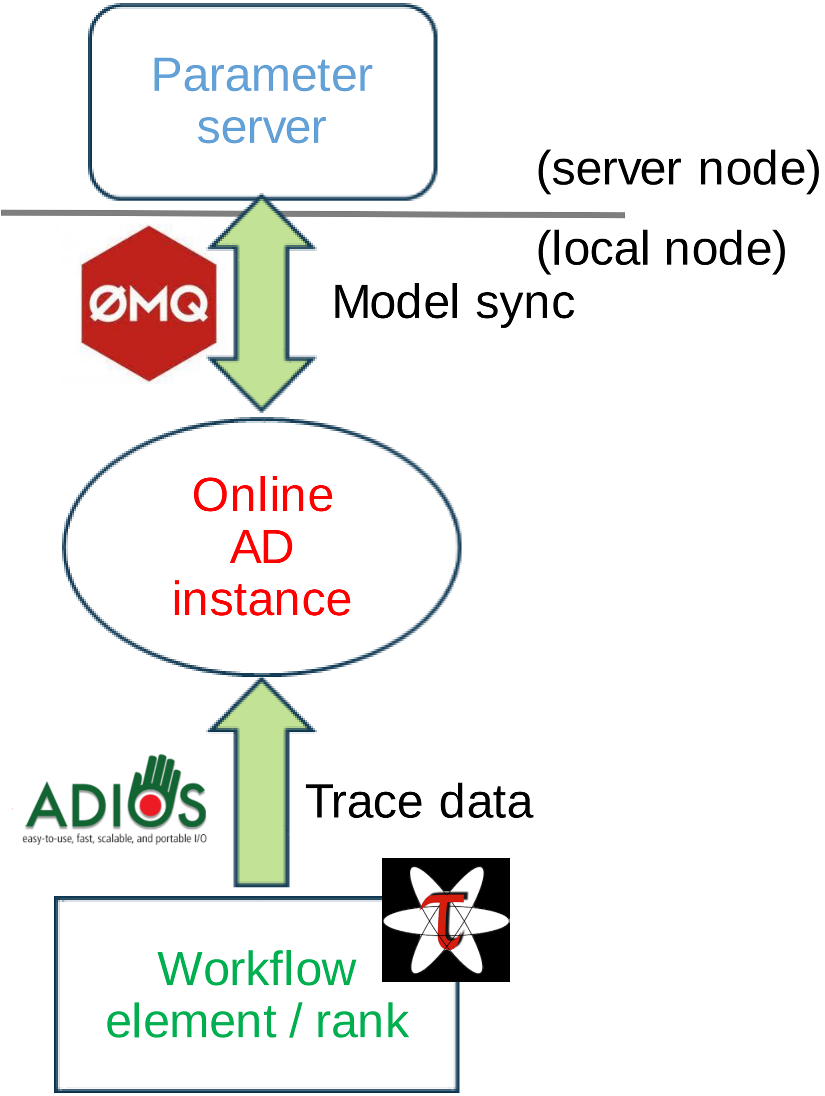
https://github.com/CODARcode/Chimbuko



- Chimbuko performs real-time *in situ* analysis of trace data captured by TAU.
- All workflow component instances simultaneously analyzed by local "Online AD" processes.
- Focus on isolating anomalous behavior using ML-driven approach.
- Detailed provenance information is stored for each anomaly.
- Remaining trace data is discarded, resulting in a dramatic reduction in data volume.

EXASCALE COMPUTING PROJECT

Anomaly detection



- Trace data is obtained by TAU and piped to local OAD via ADIOS2 in batches (~1 batch / second).
- For each *function* the OAD builds a model of the executions in the batch.
- Presently model only function runtime.
- Model parameters are merged/sync'd with global model.
- Executions in batch are then analyzed for anomalies.
- Supported AD algorithms:
  - **Histogram-based outlier selection** (HBOS)
    - Runtime histogram generated, outliers chosen based on bin likelihood [Goldstein, Dengel,2012]
  - **Copula-based outlier detection** (COPOD)
    - Also histogram-based but utilizes empirical CDF [Li, Zhao *et al*, 2020]
  - **Gaussian model** (SSTD)
    - Executions modeled as a normal distribution
- Parameter server optimized to support thousands of OAD client instances.

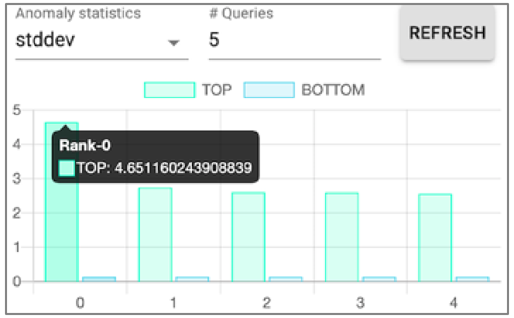
EXASCALE COMPUTING PROJECT

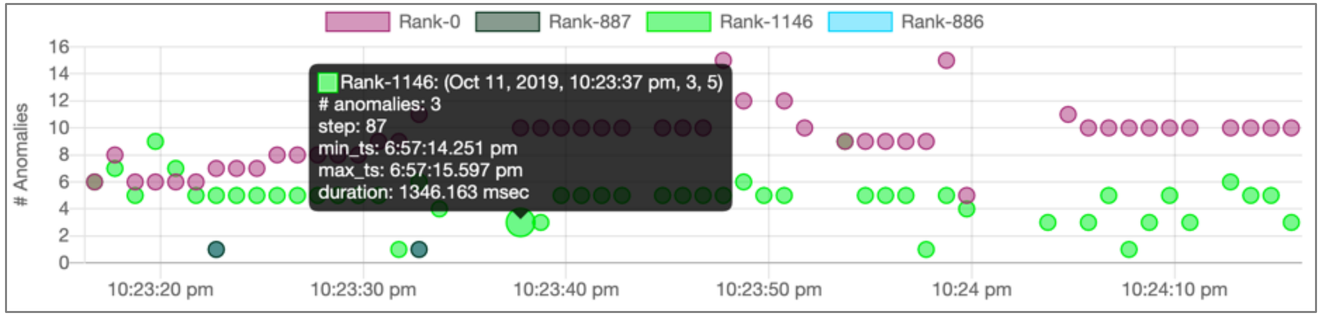
Provenance information

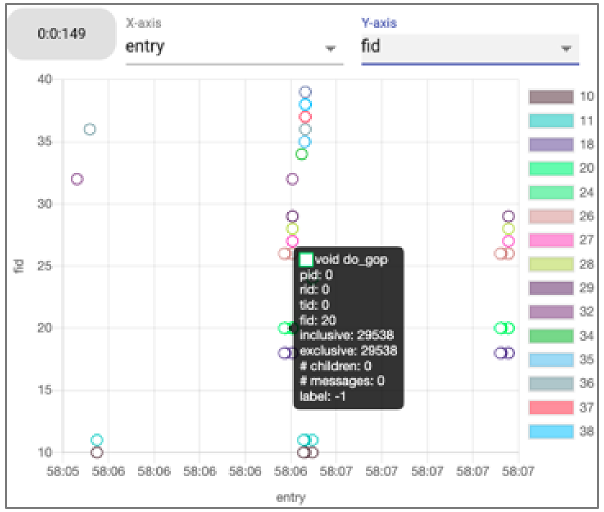

```
{
    "__id": 10,
    "algo_params": {
        "Histogram Bin Counts": [
            6,
            3
        ],
        "Histogram Bin Edges": [
            144879.999438948,
            145330.999719474,
            145782
        ]
    },
    "call_stack": [
        {
            "entry": 1646924305592505,
            "event_id": "0:0:11165",
            "exit": 1646924305738895,
            "fid": 0,
            "func": ".TAU application",
            "is_anomaly": true
        }
    ],
    "counter_events": [],
    "entry": 1646924305592505,
    "event_id": "0:0:11165",
    "event_window": {
        "comm_window": [],
        "exec_window": [
            {
                "entry": 1646924305592505,
                "event_id": "0:0:11165",
                "exit": 1646924305738895,
                "fid": 0,
                "func": ".TAU application",
                "is_anomaly": true,
                "parent_event_id": "root"
            },
            {
                "entry": 1646924305737314,
                "event_id": "0:0:11166",
                "exit": 1646924305738891,
                "fid": 290,
                "func": "[PTHREAD] execute_native_thread_routine [{thread.cc} {0, 0}]",
                "is_anomaly": false,
                "parent_event_id": "0:0:11165"
            }
        ]
    },
    "exit": 1646924305738895,
    "fid": 0,
    "func": ".TAU application",
    "gpu_location": null,
    "gpu_parent": null,
    "hostname": "node06",
    "io_step": 0,
    "io_step_tend": 1646924305738895,
    "io_step_tstart": 1646924304530378,
    "is_gpu_event": false,
    "outlier_score": 100.000111389792,
    "outlier_severity": 144813,
    "pid": 0,
    "rid": 0,
    "runtime_exclusive": 144813,
    "runtime_total": 146390,
    "tid": 4
},
```

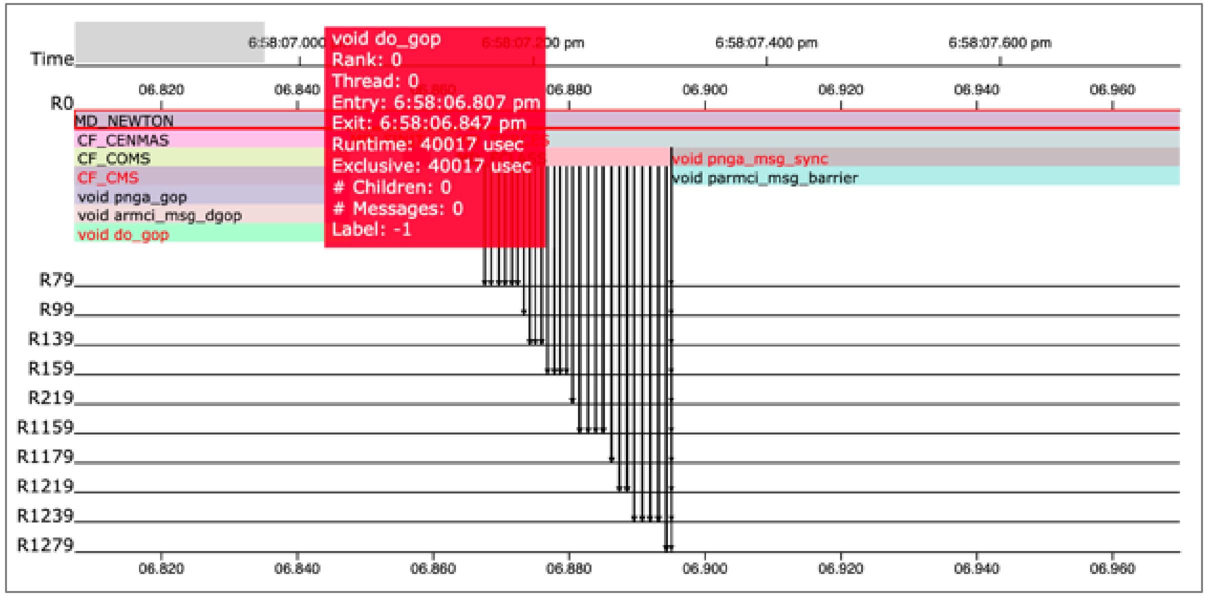

- To enable root-cause identification we must capture detailed provenance.
  - Execution parameters
    - Inclusive/exclusive runtime, timestamp, function name
  - Location information
    - Rank, device, host, thread, etc
  - Call stack information
    - Both host and device side for GPU kernel executions
  - Performance counters captured during function execution
    - PAPI counters, disk activity, GPU API-provided counters
  - MPI communication events during function execution
  - Algorithm parameters used to make outlier decision
- Data are formatted as JSON records and sent to centralized provenance database.

Chimbuko Visualization





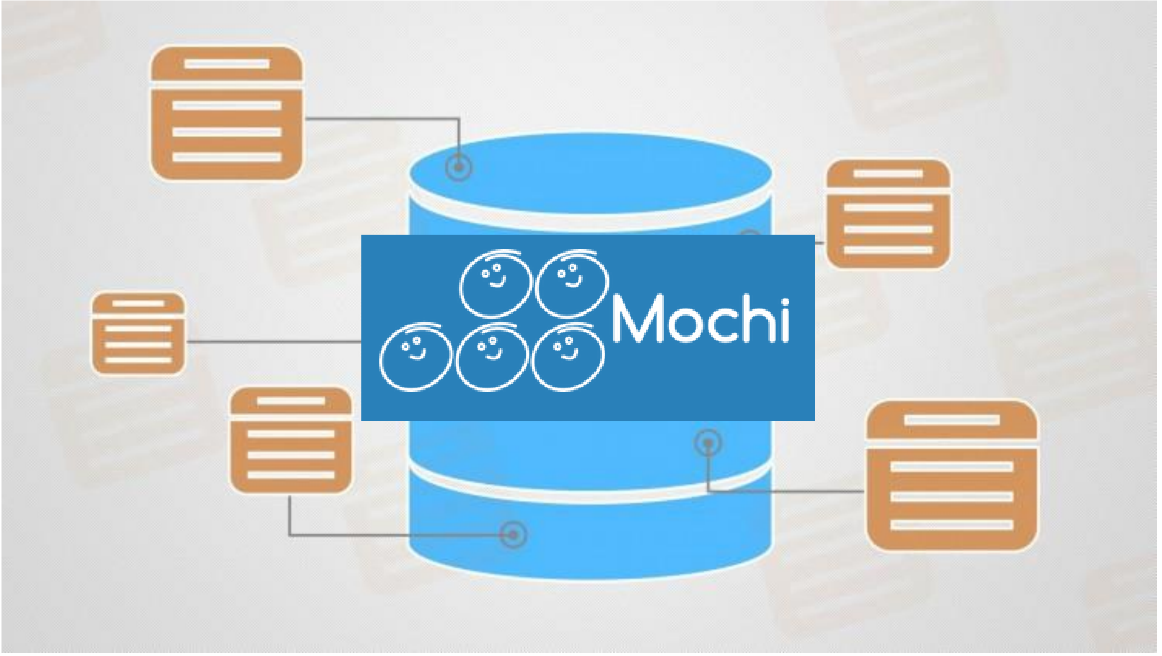




- Online visualization tool provides user overview and provDB access.
- Drill down from rank to individual anomaly
- Call stack and MPI comms visualization.

EXASCALE COMPUTING PROJECT

Provenance database

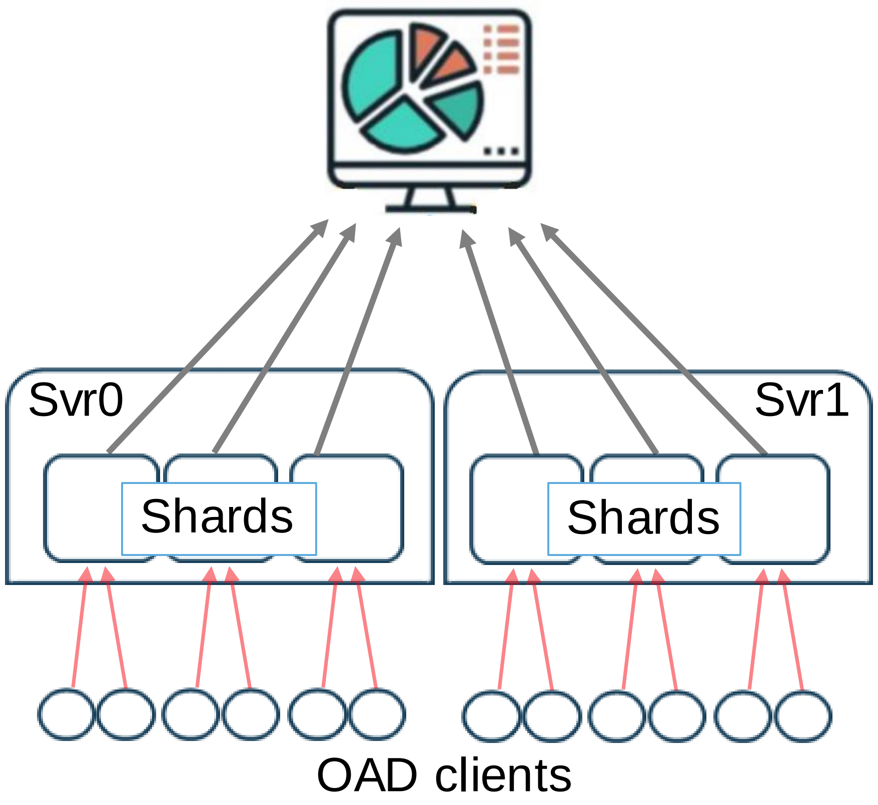


- Provenance database runs on the server node and collects provenance data from all ranks.
- Require a remote (JSON) document-store, non-relational database with:
  - Support for asynchronous stores from clients
  - Low-latency read access to support visualization.
  - Scalability to potentially thousands of simultaneous clients.
    (i.e. 1000s of records stored / s)

[https://github.com/mochi-hpc/mochi-sonata]

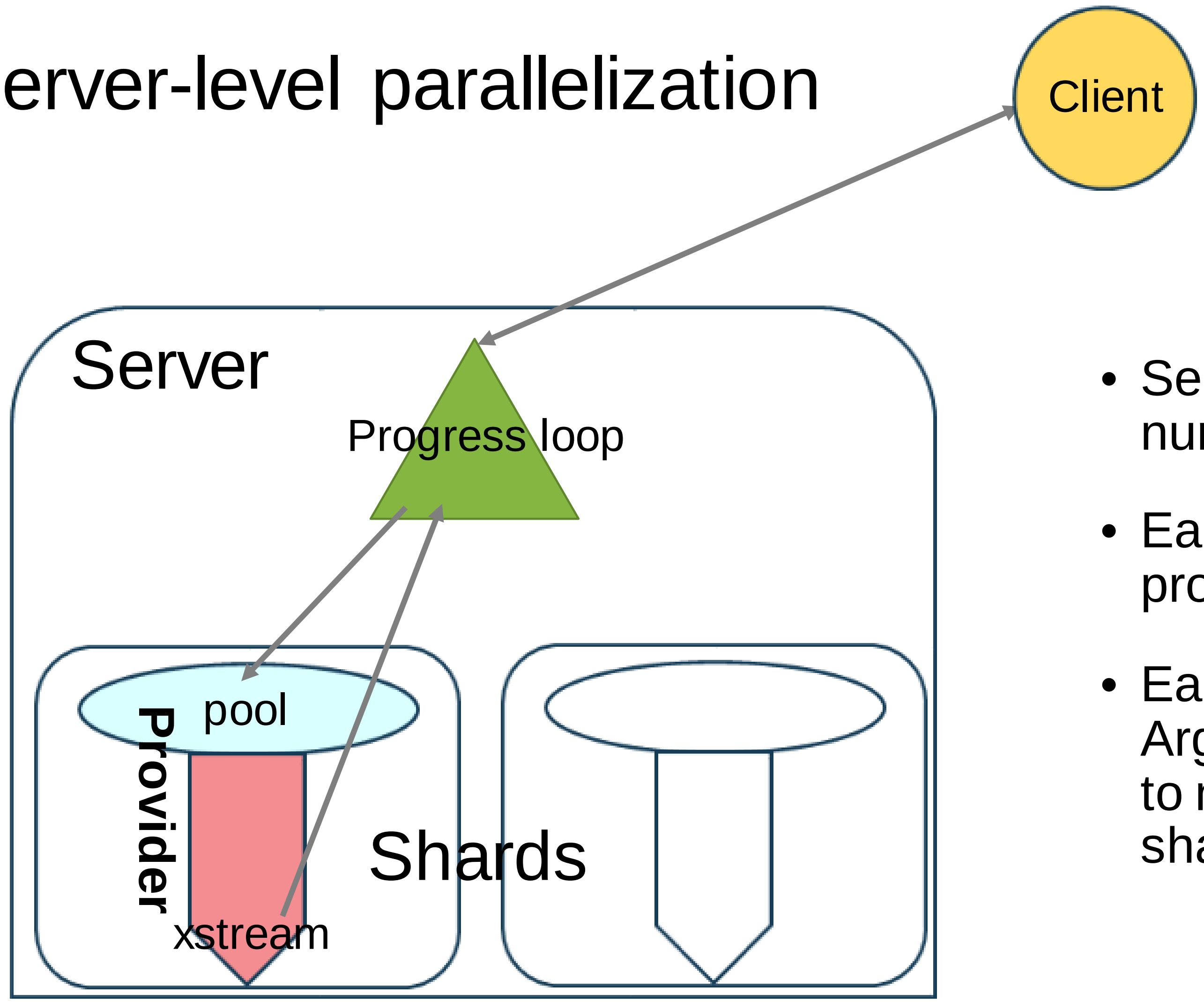
- Our implementation uses *Sonata*
  - A Mochi service codesigned by Matthieu Dorier.
  - Remote access to UnQLite database instances.
  - Jx9 query language enables arbitrary filtering.
  - C++ and Python client support.

A scalable design



- Database sharding allows for a scalable design capable of supporting large numbers of clients:
  - Clients each connect to a single shard
  - Server instances control multiple shards
  - Additional server instances can be maintained on independent resources to avoid hardware constraints
  - Visualization connects to every shard but accesses infrequent as driven by direct user interaction with frontend.

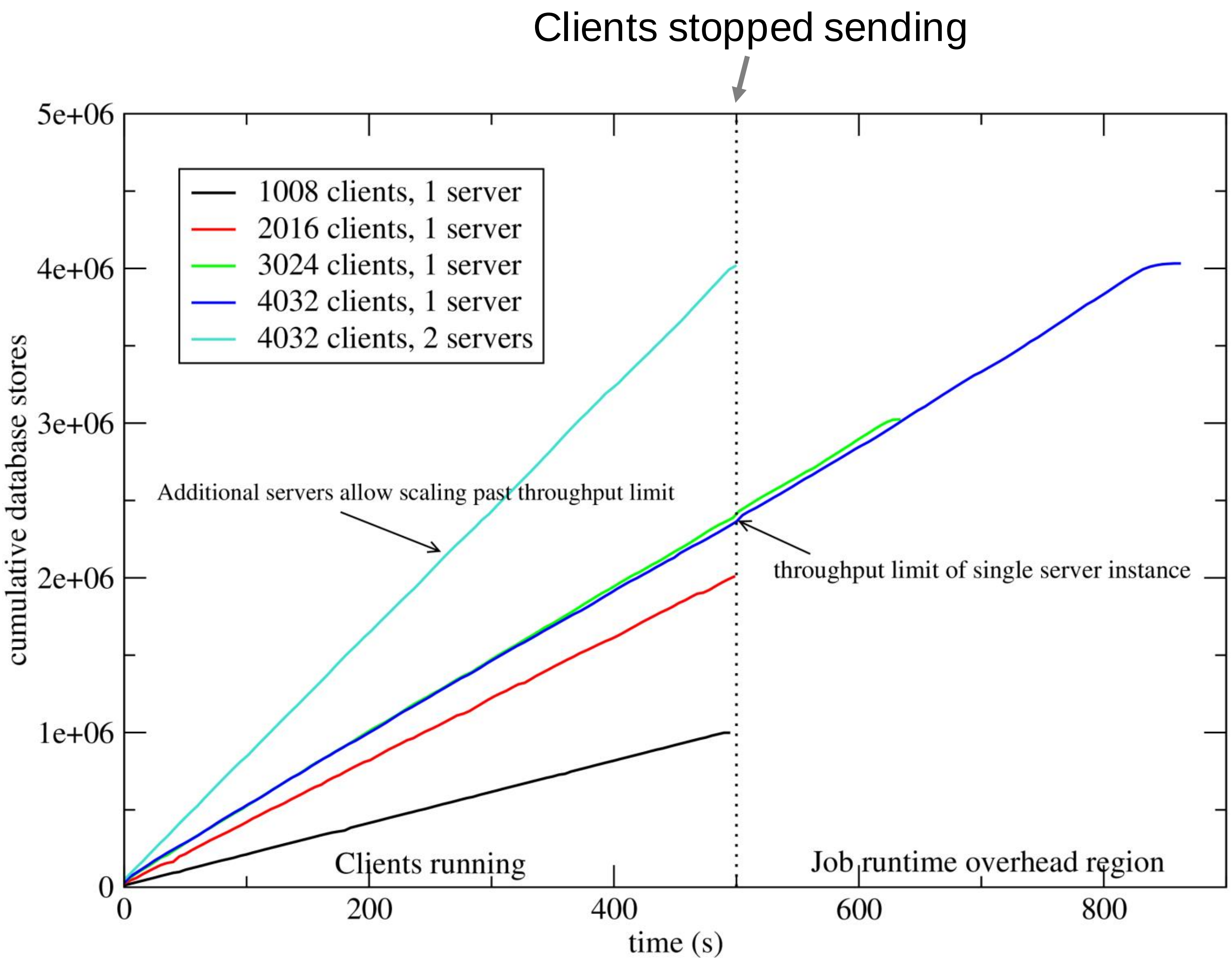
Server-level parallelization



- Server can support an arbitrary number of shards.
- Each shard is an independent Margo provider
- Each provider bound to independent Argobots execution stream and pool to minimize interference between shards.

Scalability study (Summit)

- Scalability study performed on Summit
- Assume 2 stores / second / client
- Single server demonstrated capability of supporting up to O(2500) clients
- Additional server instances allow unlimited scalability.



Mochi Yokan

[https://github.com/mochi-hpc/mochi-yokan]

- UnQLite is not the fastest database solution on the market.
  - Hacking the API to call into lower-level functionality is necessary to achieve best performance.
- Some issues encountered with stability and thread safety.
- Databases such as Facebook's RocksDB and Google's LevelDB may improve server capacity
  - Some also offer compression to reduce provDB memory footprint.
- The new Mochi "Yokan" service is an evolution of Sonata to support many different backends (including RocksDB and LevelDB)
- The Chimbuko team are working with the Mochi team to replace the Sonata implementation with Yokan.
  - Preliminary implementation complete and benchmarking is underway.

Summary

- The ECP Chimbuko tool allows for real-time performance monitoring for workflows running at-scale on HPC machines.
- The application is modeled and outliers detected using unsupervised machine learning algorithms.
- Detailed provenance information is captured and stored in a highly scalable database implemented as a Mochi Sonata microservice codesigned by the Mochi team.
- Visualization tools allow for online and offline analysis of the resulting data.
- We look forward to continued collaboration with the Mochi team!