Yandex progress on EventIndex

Alexey Artemov and Andrey Ustyuzhanin

April 16, 2014

Existing EventIndex deployment

- 10-machine HBase cluster (92 physical CPUs, 48 GB RAM/node, 80 TB HDD)
- LHCb 2012 data only
  - 1.1×10^{10} events
  - 1 TB compressed ROOT data (3.5 TB HBase table)
- Supports retrieving events by event-id (eventNumber, runNumber, stream, version)
- Stripping lines index only (1 TB HBase table)
  - Index used whenever possible ($\approx$ for rare stripping lines)
  - Other search criteria require full-scan

EventIndex product requirements

- Store LHCb 2010 — . . . data
- Store ATLAS data (which years?)
- Support queries for summary hashes
  - Add filtering by stream and version
- Support quick queries for multiple stripping lines
  - Add filtering by stream and version
- Query suggest/autocomplete
- Get the number of matching events

Indexing

Has undergone a major refactoring.

- All indexing rewritten in Java
  - Reuse common code, slightly accelerate building
- Added “dense” index type for summary hashes
- Added configuration file
  - Easily select fields for indexing via regexp
  - Use key or key-value index
  - Select index type (currently “sparse” or “dense”)
  - Select formatting
- Added indexing tests (launchable from python)

Search: infrastructure

Introduce distributed search infrastructure having multiple components launched as separate processes.

- Masters
  - Communicate with clients, cache information for quick retrieval
  - Never executes heavy IO jobs (1-2 seconds requests)
  - Typically #masters = #django workers
- IPC handlers
  - Run in background, parse queries, determine query types
  - Distribute load across workers
  - Typically #handlers = 2×#masters
- Workers
  - Do heavy IO (including table full-scans)
  - 6–10 workers per RegionServer
- Comminucate via TCP sockets using 0MQ

Search: query syntax

- Use `HAS` operator for sparse factors
  (e.g. `HAS StrippingXiccXiccPlusToLcKPiWCDecision`)
- Use `factor OP value` for numeric dense factors
  (e.g. `nRich1Hits >= 1000`)
- Combine conditions using `AND` and `OR` operators
- Complex queries possible with parentheses
  (e.g. `(HAS s1 AND HAS s2) OR HAS s3`)
- Builds query parse tree from string
- Determines query kind by looking for predefined query patterns
  (currently a single sparse factor only)

Search: some performance

Query	Events	Time, sec.
StrippingDstarUPB02DstDstBeauty2CharmLineDecision	7	1
StrippingDstarUPB2DstDstKBeauty2CharmLineDecision	70	1
Strippingb2DstarMuXKsKs_DDDDCharmFromBSemiLineDecision	1313	1
StrippingB02DstDstKSWSLLBeauty2CharmLineDecision	11 474	1
StrippingDstarUPB2DstD0Beauty2CharmLineDecision AND stripping=20r1	38 141	15
StrippingDstarUPB2DstD0Beauty2CharmLineDecision	112 558	18

ATLAS indexing overview

- Index data 2010 – 2020
- Obtain raw data from GRID into HDFS at Yandex
  - Extend 'Mask' fields from Mask COMA database of chains
  - JSON format preferred
- Index raw data
  - Consider development of new index types
- Load-test indexed data

ATLAS indexing details

- Index triggers (EFPassedTrigMask, L1PassedTrigMaskTAP, L1PassedTrigMaskTAV, L1PassedTrigMaskTBP, L2PassedTrigMask)
- Index all triggers jointly
- Index runnumber + triggers
- Index runnumber + all triggers jointly
- Index GUIDs (indexes StreamAOD_ref_1, StreamESD_ref_1, StreamRAW_ref_1)
- Add support for automatic index selection depending on user query

ATLAS milestones

- We will start in June 2014
- Data acquisition and upload — 1 week
- Indexing using existing index types (8 indexes) + index verification — 1 week
- Development of new index types (6 indexes) + indexing + index verification — 3 weeks
  - Index to store dense and sparse factors together (e.g. sparse factors + runNumber) — 1 week
  - (possibly) Index types for string factors — 1 week
  - Testing, indexing and index verification — 1 week
- Load testing — 1 week
- Total: 6 weeks