A Hybrid System for Scientific Data Processing with QoS

Current Status Report

Noah Watkins
jayhawk@cs.ucsc.edu
Outline

- Motivation
- Problem Space
- Current Status
- Looking Forward
- Questions
Science Data: Big and Important

- Scientific data is growing rapidly in volume
- Disciplines are becoming data centric
- Dependence on data is increasing too
  - New/bigger data == new questions
Science Data: Support

- Institutions do not have the resources to support massive data sets
  - Storage and analysis (petabyte: $$$)
- Data integration compounds the problem
- What's needed is shared storage and analysis resources for scientific data
Case Study: LSST

- Large Synoptic Survey Telescope
  - Giant telescope based in Chile
- Cool factor
  - Real-time alert generation, world-wide nightly data transfers, 24/7 data analysis, public access
- Planned 10 year life (2018 – 2028)
  - 3 GB image / 15 seconds / night (0.5 PB / month)
  - Raw and derived metadata kept in catalog
The LSST Catalog

- Massive metadata catalog
  - Cataloging 50 billion stars
  - 10x trillion observations
- Vast majority of analysis is metadata-only
- Wide variety of queries and users
  - Professional and amateur astronomers
  - Academic and general public
The LSST Catalog Database

<table>
<thead>
<tr>
<th>Example Table</th>
<th>Data Release 11 (2028)</th>
<th>Row Count</th>
<th>10 Year Growth</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object</td>
<td>&gt; 100 TB</td>
<td>40 billion</td>
<td>3x</td>
</tr>
<tr>
<td>Source</td>
<td>&gt; 1 PB</td>
<td>5 trillion</td>
<td>20x</td>
</tr>
<tr>
<td>ForcedSource</td>
<td>~ 1 PB</td>
<td>30 trillion</td>
<td>50x</td>
</tr>
</tbody>
</table>

- Metadata stored as relational data
- Raw images stored in file system
- Multiple versions of data
  - Produced from different algorithms
  - Estimates as high as 400 trillion rows
LSST Queries and QoS

<table>
<thead>
<tr>
<th>Query Class (Data Vol)</th>
<th>Latency</th>
<th>Concurrency (Load)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Low</td>
<td>&lt; 10 sec</td>
<td>50</td>
</tr>
<tr>
<td>High</td>
<td>&lt; 1 hour</td>
<td>20</td>
</tr>
<tr>
<td>Super High</td>
<td>&lt; 1 week</td>
<td>1</td>
</tr>
</tbody>
</table>

- **Low volume**
  - Object retrieval using index
- **High volume**
  - Aggregate and search with table scan
- **Super High**
  - KNN, joins, and correlations
Basic Requirements

- Allow shared access to data and analysis
  - Massive data → distributed solutions
- Handles diverse set of QoS requirements
- Efficient and incrementally scalable
  - LSST will not allocate resources in 2018 for 2028
- Adaptable to many different data models
Why is it so hard?

- Lack of *coordination* in the shared system
  - Induced random IO degrades performance
- Best-effort assumptions
  - *Many* system components
- Lack of *isolation*
  - Interference hurts predictability
- No single data model / file format
## Problem Space

<table>
<thead>
<tr>
<th>Data Processing</th>
<th>Performance Management</th>
<th>Data Models</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
# Problem Space

<table>
<thead>
<tr>
<th>Data Processing</th>
<th>Performance Management</th>
<th>Data Models</th>
</tr>
</thead>
<tbody>
<tr>
<td>Indexing for low-latency query evaluation</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Problem Space

<table>
<thead>
<tr>
<th>Data Processing</th>
<th>Performance Management</th>
<th>Data Models</th>
</tr>
</thead>
<tbody>
<tr>
<td>Indexing for low-latency query evaluation</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Efficiency at high loads – shared-table scans</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Problem Space

<table>
<thead>
<tr>
<th>Data Processing</th>
<th>Performance Management</th>
<th>Data Models</th>
</tr>
</thead>
<tbody>
<tr>
<td>Indexing for low-latency query evaluation</td>
<td>Predictability and isolation</td>
<td></td>
</tr>
<tr>
<td>Efficiency at high loads – shared-table scans</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Problem Space

<table>
<thead>
<tr>
<th>Data Processing</th>
<th>Performance Management</th>
<th>Data Models</th>
</tr>
</thead>
</table>
| Indexing for low-latency query evaluation | Predictability and isolation | – **CPU**  
– Network  
– I/O  
– Buffer-cache |
Problem Space

<table>
<thead>
<tr>
<th>Data Processing</th>
<th>Performance Management</th>
<th>Data Models</th>
</tr>
</thead>
<tbody>
<tr>
<td>Indexing for low-latency query evaluation</td>
<td>Predictability and isolation</td>
<td>Relational</td>
</tr>
<tr>
<td>Efficiency at high loads</td>
<td>– CPU</td>
<td></td>
</tr>
<tr>
<td>– shared-table scans</td>
<td>– Network</td>
<td></td>
</tr>
<tr>
<td></td>
<td>– I/O</td>
<td></td>
</tr>
<tr>
<td></td>
<td>– Buffer-cache</td>
<td></td>
</tr>
</tbody>
</table>
# Problem Space

<table>
<thead>
<tr>
<th>Data Processing</th>
<th>Performance Management</th>
<th>Data Models</th>
</tr>
</thead>
<tbody>
<tr>
<td>Indexing for low-latency query evaluation</td>
<td>Predictability and isolation</td>
<td>Relational</td>
</tr>
<tr>
<td></td>
<td>– CPU</td>
<td>Array</td>
</tr>
<tr>
<td></td>
<td>– Network</td>
<td>Mesh</td>
</tr>
<tr>
<td></td>
<td>– I/O</td>
<td>Unstructured</td>
</tr>
<tr>
<td>Efficiency at high loads – shared-table scans</td>
<td>– Buffer-cache</td>
<td></td>
</tr>
</tbody>
</table>

Complete Solution

<table>
<thead>
<tr>
<th>Data Processing</th>
<th>Performance Management</th>
<th>Data Models</th>
</tr>
</thead>
<tbody>
<tr>
<td>Indexing for low-latency query evaluation</td>
<td>Predictability and isolation</td>
<td>Relational</td>
</tr>
<tr>
<td>Efficiency at high loads</td>
<td>– CPU</td>
<td>Array</td>
</tr>
<tr>
<td>– shared-table scans</td>
<td>– Network</td>
<td>Mesh</td>
</tr>
<tr>
<td></td>
<td>– I/O</td>
<td>Unstructured</td>
</tr>
<tr>
<td></td>
<td>– Buffer-cache</td>
<td></td>
</tr>
</tbody>
</table>

Hybrid Design
# Complete Solution

<table>
<thead>
<tr>
<th>Data Processing</th>
<th>Performance Management</th>
<th>Data Models</th>
</tr>
</thead>
<tbody>
<tr>
<td>Indexing for low-latency query evaluation</td>
<td>Predictability and isolation</td>
<td>Relational</td>
</tr>
<tr>
<td>Efficiency at high loads</td>
<td>– CPU</td>
<td>Array</td>
</tr>
<tr>
<td>– shared-table scans</td>
<td>– Network</td>
<td>Mesh</td>
</tr>
<tr>
<td>Hybrid Design</td>
<td>– I/O</td>
<td>Unstructured</td>
</tr>
<tr>
<td>End-to-End QoS</td>
<td>– Buffer-cache</td>
<td></td>
</tr>
</tbody>
</table>
## Complete Solution

<table>
<thead>
<tr>
<th>Data Processing</th>
<th>Performance Management</th>
<th>Data Models</th>
</tr>
</thead>
<tbody>
<tr>
<td>Indexing for low-latency query evaluation</td>
<td>Predictability and isolation</td>
<td>Relational</td>
</tr>
<tr>
<td>Efficiency at high loads – shared-table scans</td>
<td>– CPU</td>
<td>Array</td>
</tr>
<tr>
<td></td>
<td>– Network</td>
<td>Mesh</td>
</tr>
<tr>
<td></td>
<td>– I/O</td>
<td>Unstructured</td>
</tr>
<tr>
<td></td>
<td>– Buffer-cache</td>
<td><strong>Applicable to Scientific Data</strong></td>
</tr>
</tbody>
</table>

*Hybrid Design*  

*End-to-End QoS*
# Current Status

<table>
<thead>
<tr>
<th>Data Processing</th>
<th>Performance Management</th>
<th>Data Models</th>
</tr>
</thead>
<tbody>
<tr>
<td>Indexing for low-latency query evaluation</td>
<td>Predictability and isolation</td>
<td>Relational</td>
</tr>
<tr>
<td>Efficiency at high loads</td>
<td>– CPU</td>
<td>Array</td>
</tr>
<tr>
<td>– shared-table scans</td>
<td>– Network</td>
<td>Mesh</td>
</tr>
<tr>
<td></td>
<td>– I/O</td>
<td>Unstructured</td>
</tr>
<tr>
<td></td>
<td>– Buffer-cache</td>
<td></td>
</tr>
</tbody>
</table>

- Real-world *relational* data from LSST
- *Shared-scans* easier to model than random IO
- Existing work on *I/O* performance isolation
- How can these three components function together?
Shared-table Scans

- Query Q1:
  
  ```sql
  SELECT SUM(attr1)
  FROM T1;
  ```
Shared-table Scans

• Query Q1:
  SELECT SUM(attr1)
  FROM T1;

• Full scan of table T1
Shared-table Scans

- Query Q1:
  
  ```sql
  SELECT SUM(attr1) 
  FROM T1;
  ```

- Full scan of table T1

- Select all rows from T1
Shared-table Scans

- Query Q1:
  `SELECT SUM(attr1) FROM T1;`
- Full scan of table T1
- Select all rows from T1
- Calculate sum
Shared-table Scans

- Query Q1:
  
  ```sql
  SELECT SUM(attr1)
  FROM T1;
  ```

- Query Q2:
  
  ```sql
  SELECT SUM(attr2)
  FROM T1 WHERE attr2 < 10;
  ```
Shared-table Scans

- Query Q1:
  
  ```sql
  SELECT SUM(attr1)
  FROM T1;
  ```

- Query Q2:
  
  ```sql
  SELECT SUM(attr2)
  FROM T1 WHERE attr2 < 10;
  ```

- Both queries can share the work of reading data off disk
  - Shared-table scan
Shared-table Scans

- **Query Q1:**
  
  ```sql
  SELECT SUM(attr1)
  FROM T1;
  ```

- **Query Q2:**
  
  ```sql
  SELECT SUM(attr2)
  FROM T1 WHERE attr2 < 10;
  ```

- **Query Q3 (hash-join):**
  
  ```sql
  T1 ⋈ T2
  ```
Shared-table Scans

- Query Q1:
  ```sql
  SELECT SUM(attr1)
  FROM T1;
  ```

- Query Q2:
  ```sql
  SELECT SUM(attr2)
  FROM T1 WHERE attr2 < 10;
  ```

- Query Q3 (hash-join): $T_1 \bowtie T_2$
  - Phase 1: Scan $T_2$ (construct hash table)
Shared-table Scans

- Query Q1:
  SELECT SUM(attr1)
  FROM T1;

- Query Q2:
  SELECT SUM(attr2)
  FROM T1 WHERE attr2 < 10;

- Query Q3 (hash-join): $T_1 \bowtie T_2$
  - Phase 1: Scan T2 (construct hash table)
  - Phase 2: Scan T1 (probe hash table)
QoS for Shared-table Scans

• Associate latency requirement with query
  • Query $\rightarrow$ (Expression, Latency)
  • Example: Q1 $\rightarrow$ (SELECT SUM... , 1 hr.)

• Goals
  • Hard RT latency guarantees
  • Soft RT/Best effort latency guarantees
  • Admission control
    - Estimated completion
QoS for Shared-table Scans

- Simplifying assumptions
  - Single node, one hard drive
QoS for Shared-table Scans

- Simplifying assumptions
  - Single node, one hard drive
  - Dominate cost: I/O
    - Bandwidth reservations with isolation
    - Existing technology: Fahrrad

1. https://systems.soe.ucsc.edu/node/308#proj3
QoS for Shared-table Scans

- Simplifying assumptions
  - Single node, one hard drive
  - Dominate cost: I/O
    - Bandwidth reservations with isolation
    - Existing technology: Fahrrad
  - No other costs
    - CPU, memory, network buffer-cache, etc...
    - Important next steps

1. https://systems.soe.ucsc.edu/node/308#proj3
Example: Shared Scan QoS

- **Query Q1**: `SELECT SUM(attr1) FROM T1;` **Latency: L1**
Example: Shared Scan QoS

- **Query Q1:** `SELECT SUM(attr1) FROM T1;`\[Latency: \textbf{L1}\]
- **Query Q2:** `SELECT SUM(attr2) FROM T1 WHERE attr2 < 10;`\[Latency: \textbf{L2}\]

![Diagram of shared scan QoS](image)
Example: Shared Scan QoS

- Query Q1: \( \text{SELECT} \ \text{SUM}(\text{attr1}) \ \text{FROM} \ T1; \) Latency: \( L1 \)
- Query Q2: \( \text{SELECT} \ \text{SUM}(\text{attr2}) \ \text{FROM} \ T1 \ \text{WHERE} \ \text{attr2} < 10; \) \( L2 \)
- Query Q3: \( T1 \bowtie T2; \) \( L3 \rightarrow L3' + L3'' \)
  - Phase 1: Scan T2 (construct hash table) \( L3' \)
  - Phase 2: Scan T1 (probe hash table) \( L3'' \)
Example: Shared Scan QoS

- Query Q1: `SELECT SUM(attr1) FROM T1;`  Latency: $L1$
- Query Q2: `SELECT SUM(attr2) FROM T1 WHERE attr2 < 10;`  $L2$
- Query Q3: $T1 \bowtie T2;  L3 \rightarrow L3' + L3''$
  - Phase 1: Scan T2 (construct hash table) $L3'$
  - Phase 2: Scan T1 (probe hash table) $L3''$
- Bandwidth needs for scan of T2?
  - $|T2| / L3'$ (e.g. MB/s)
Example: Shared Scan QoS

- Query Q1: \( \text{SELECT SUM(attr1) FROM T1}; \) \text{ Latency: } L1
- Query Q2: \( \text{SELECT SUM(attr2) FROM T1 WHERE attr2 < 10}; \) \text{ L2}
- Query Q3: \( T1 \bowtie T2; \) \( L3 \rightarrow L3' + L3'' \)
  - Phase 1: Scan T2 (construct hash table) \( L3' \)
  - Phase 2: Scan T1 (probe hash table) \( L3'' \)
- Bandwidth needs for scan of T2?
  - \( |T2| / L3' \) (e.g. MB/s)
- Bandwidth needs for scan of T1?
  - \( |T1| / \text{min}(L1, L2, L3'') \)
Shared Scan State Space

- Bandwidth need for scan of T1?
  - $|T1| / \min(L1, L2, L3'')$

- Avoid over provisioning
  - Q3/Phase2 strictly after Q3/Phase1
    - Over provision T1 scan in Phase 1
    - Admission control knows about all available bandwidth

- Q1 and Q2 may independently start and finish
Shared Scan State Space

- Bandwidth need for scan of T1?
  - $|T1| / \min(L1, L2, L3'')$

- Avoid over provisioning
  - Q3/Phase2 \textit{strictly after} Q3/Phase1
    - Over provision T1 scan in Phase 1
    - Admission control knows about all available bandwidth
  - Q1 and Q2 may independently start and finish

- State space $\rightarrow$ valid scan combinations
Shared Scan State Space

- Bandwidth need for scan of T1?
  - $|T1| / \min(L1, L2, L3)$
- Avoid over provisioning
  - Q3/Phase2 *strictly after* Q3/Phase1
    - Over provision T1 scan in Phase 1
    - Admission control knows about all available bandwidth
- Q1 and Q2 may independently start and finish
- State space → valid scan combinations
Shared Scan State Space

- Bandwidth need for scan of T1?
  - $|T1| \div \min(L1, L2, L3")$

- Avoid over provisioning
  - Q3/Phase2 strictly after Q3/Phase1
    - Over provision T1 scan in Phase 1
    - Admission control knows about all available bandwidth
  - Q1 and Q2 may independently start and finish

- State space $\rightarrow$ valid scan combinations
Shared Scan State Space

- Bandwidth need for scan of T1?
  - \|T1\| / \min(L1, L2, L3)"
- Avoid over provisioning
  - Q3/Phase2 \textit{strictly after} Q3/Phase1
    - Over provision T1 scan in Phase 1
    - Admission control knows about all available bandwidth
- Q1 and Q2 may independently start and finish
- State space $\rightarrow$ valid scan combinations
Shared Scan State Space

- Bandwidth need for scan of T1?
  - $|T1| / \min(L1, L2, L3")$
- Avoid over provisioning
  - Q3/Phase2 *strictly after* Q3/Phase1
    - Over provision T1 scan in Phase 1
    - Admission control knows about all available bandwidth
- Q1 and Q2 may independently start and finish
- State space $\rightarrow$ valid scan combinations
Shared Scan State Space

- Bandwidth need for scan of T1?
  - $|T1| / \min(L1, L2, L3")$
- Avoid over provisioning
  - Q3/Phase2 strictly after Q3/Phase1
    - Over provision T1 scan in Phase 1
    - Admission control knows about all available bandwidth
- Q1 and Q2 may independently start and finish
- State space → valid scan combinations
Shared Scan State Space

- Bandwidth need for scan of T1?
  - $|T1| / \min(L1, L2, L3'')$
- Avoid over provisioning
  - Q3/Phase2 strictly after Q3/Phase1
    - Over provision T1 scan in Phase 1
    - Admission control knows about all available bandwidth
- Q1 and Q2 may independently start and finish
- State space → valid scan combinations

\[
\begin{align*}
\{\} \\
\{\text{T1:Q1}\} \\
\{\text{T1:Q2}\} \\
\{\text{T1:Q3}\} \\
\{\text{T1:Q2, T1:Q1}\}
\end{align*}
\]
Shared Scan State Space

- Bandwidth need for scan of T1?
  - $|T1| / \min(L1, L2, L3)$
- Avoid over provisioning
  - Q3/Phase2 strictly after Q3/Phase1
    - Over provision T1 scan in Phase 1
    - Admission control knows about all available bandwidth
  - Q1 and Q2 may independently start and finish
- State space $\rightarrow$ valid scan combinations

The state space depends on
- Current workload (query set)
- Implementation (e.g. hash-join)
Using the State Space

• Making worst-case, static reservations
  • Without over-provisioning
Using the State Space

- Making worst-case, static reservations
  - Without over-provisioning
Using the State Space

- Making worst-case, static reservations
  - Without over-provisioning
- Admission control
  - Detecting QoS violations for arriving queries

Cannot meet QoS req. of new query
Using the State Space

- Making worst-case, static reservations
  - Without over-provisioning
- Admission control
  - Detecting QoS violations for arriving queries
  - Constructing new schedules
Using the State Space

- Making worst-case, static reservations
  - Without over-provisioning
- Admission control
  - Detecting QoS violations for arriving queries
  - Constructing new schedules
- Soft real time analysis
  - Probabilistic states
Current Focus

• Designing formalisms
  • Describe IO behavior of query sets
  • Express admission control policies

• Baseline and model validation experiment design
## Next Steps

<table>
<thead>
<tr>
<th>Data Processing</th>
<th>Performance Management</th>
<th>Data Models</th>
</tr>
</thead>
<tbody>
<tr>
<td>Indexing for low-latency query evaluation</td>
<td>Predictability and isolation</td>
<td>Relational</td>
</tr>
<tr>
<td></td>
<td>– CPU</td>
<td>Array</td>
</tr>
<tr>
<td></td>
<td>– Network</td>
<td>Mesh</td>
</tr>
<tr>
<td></td>
<td>– I/O</td>
<td>Unstructured</td>
</tr>
<tr>
<td>Efficiency at high loads</td>
<td></td>
<td></td>
</tr>
<tr>
<td>– shared-table scans</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Validating the current model
## Next Steps

<table>
<thead>
<tr>
<th>Data Processing</th>
<th>Performance Management</th>
<th>Data Models</th>
</tr>
</thead>
<tbody>
<tr>
<td>Indexing for low-latency query evaluation</td>
<td>Predictability and isolation</td>
<td>Relational</td>
</tr>
<tr>
<td>Efficiency at high loads</td>
<td>– CPU</td>
<td>Array</td>
</tr>
<tr>
<td>– shared-table scans</td>
<td>– Network</td>
<td>Mesh</td>
</tr>
<tr>
<td></td>
<td>– I/O</td>
<td>Unstructured</td>
</tr>
</tbody>
</table>
# Next Steps

## Data Processing
- Indexing for low-latency query evaluation
- Efficiency at high loads
  - shared-table scans

## Performance Management
- Predictability and isolation
  - CPU
  - Network
  - I/O
  - Buffer-cache

## Data Models
- Relational
- Array
- Mesh
- Unstructured

- Validating the current model
- Extending model to include index lookup
  - Random + sequential IO
- Multiple disks
## Future Work

<table>
<thead>
<tr>
<th>Data Processing</th>
<th>Performance Management</th>
<th>Data Models</th>
</tr>
</thead>
<tbody>
<tr>
<td>Indexing for low-latency query evaluation</td>
<td>Predictability and isolation</td>
<td>Relational</td>
</tr>
<tr>
<td>Efficiency at high loads</td>
<td>– CPU</td>
<td>Array</td>
</tr>
<tr>
<td>– shared-table scans</td>
<td>– Network</td>
<td>Mesh</td>
</tr>
<tr>
<td></td>
<td>– I/O</td>
<td>Unstructured</td>
</tr>
<tr>
<td></td>
<td>– Buffer-cache</td>
<td></td>
</tr>
</tbody>
</table>

- Data structures for predictable index behavior
## Future Work

<table>
<thead>
<tr>
<th>Data Processing</th>
<th>Performance Management</th>
<th>Data Models</th>
</tr>
</thead>
<tbody>
<tr>
<td>Indexing for low-latency query evaluation</td>
<td>Predictability and isolation</td>
<td>Relational</td>
</tr>
<tr>
<td>Efficiency at high loads – shared-table scans</td>
<td>– CPU</td>
<td>Array</td>
</tr>
<tr>
<td></td>
<td>– Network</td>
<td></td>
</tr>
<tr>
<td></td>
<td>– I/O</td>
<td></td>
</tr>
<tr>
<td></td>
<td>– Buffer-cache</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Mesh</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Unstructured</td>
</tr>
</tbody>
</table>

- Data structures for predictable index behavior
- Using knowledge of data model semantics to increase predictability
Future Work

- Data structures for predictable index behavior
- Using knowledge of data model semantics to increase predictability
- Efficient scanning of diverse file layouts
Future Work

<table>
<thead>
<tr>
<th>Data Processing</th>
<th>Performance Management</th>
<th>Data Models</th>
</tr>
</thead>
<tbody>
<tr>
<td>Indexing for low-latency query evaluation</td>
<td>Predictability and isolation</td>
<td>Relational</td>
</tr>
<tr>
<td>Efficiency at high loads</td>
<td>– CPU</td>
<td>Array</td>
</tr>
<tr>
<td>– shared-table scans</td>
<td>– Network</td>
<td>Mesh</td>
</tr>
<tr>
<td></td>
<td>– I/O</td>
<td>Unstructured</td>
</tr>
<tr>
<td></td>
<td>– Buffer-cache</td>
<td></td>
</tr>
</tbody>
</table>

- Data structures for predictable index behavior
- Using knowledge of data model semantics to increase predictability
- Efficient scans of diverse file layouts
- Generalizing for distributed end-to-end QoS
Conclusion

- Problem space, current status
- Future directions

- Questions and comments welcome!
- Noah Watkins
  - http://www.cs.ucsc.edu/~jayhawk
  - jayhawk@cs.ucsc.edu