Towards Physical Design Management in Storage Systems

Kathryn Dahlgren[1], **Jeff LeFevre[1]**, Ashay Shirwadkar[2], Ken Iizawa[3], Aldrin Montana[1], Peter Alvaro[1], Carlos Maltzahn[1]

UC Santa Cruz[1], UC Riverside[2], Fujitsu Labs Ltd.[3]

PDSW'19

The Problem

- Rapid evolution of new storage devices and architectures
  - Has significant impact on the software stack
- Demise of Moore's Law may result in performance improvements driven mainly by architecture changes
  - Previous improvements required little change to stacks
- Can be difficult to take full advantage of new device-specific functionality without significant application changes
  - Frequent changes to stacks quickly becomes unaffordable
- Feasible stacks will need to move **device-specific functionality** closer to devices

Device-specific functionality

- One example is **physical design** (data management community)
  - Refers to physical data model and secondary data structures
  - Also includes how physical data is mapped to storage abstractions (files, blocks, objects)
- Physical design transformations can have significant perf benefits
  - Makes **assumptions** about underlying storage devices
    - e.g., relative performance of sequential vs. random access
- Currently, physical design resides with the application or DBA
  - Optimizes for a workload with specific (fairly static) hardware

More Complications...

- Storage hierarchies are deepening - *multiple tiers*
  - Spinning media, flash, non-volatile memory
- Dynamic movement of data between tiers
  - Benefits of a physical design may depend on which tier the data currently resides
- Physical design assumptions might not adequately reflect
  - New devices, heterogeneity of devices, or changing architectures and workloads

Our Approach

- Physical design should be managed in storage systems
  - Isolate applications and middleware from the impact of storage architectural changes
- We identify two key enabling technologies
  1. Emerging computational storage makes it possible to carry out some data processing in storage
  2. Embedding of fast serialization libraries such as Google Flatbuffers and Apache Arrow in storage allows storing and transforming structured data transparently to the application

Physical design and computational storage

- Physical design management in storage systems leverages computational storage but is not subsumed by it
  - Transformations are performed by computational storage layer but are **orchestrated** by storage clients
- Extends computational storage with physical data transformation
  - Including data formats, data orientations, data (re-)partitioning, data indexing
  - Can be on-the-fly or out-of-band
  - Can be incremental...

Orchestration of Transformations

- Transformations are beneficial but potentially long-running
  - Orchestration is an interesting research area
- Transformation plan execution space is large
  - Even given a source and target physical design
- May prefer to create benefit as quickly as possible
  - What intermediate steps to benefit production workloads?
  - What resources to dedicate for plan execution?

How do applications take advantage?

- Connect the application to new storage capabilities
  - Physical design in storage requires changes to layers above
  - Data access libraries, data management systems
- Data management systems already have **external table** facilities
  - Foreign tables, external data source, many others
- Offloads management of data to an external system
  - Offloading to distributed object storage may increase scalability
- Scientific file formats such as HDF5 have external facilities
  - Virtual Object Layer

Contributions

- Introduce and make the case for physical design management in storage systems
- Identification of research challenges
- Provide a prototype implementation in Ceph object storage
- Initial experiments at a replicable scale using Cloudlab

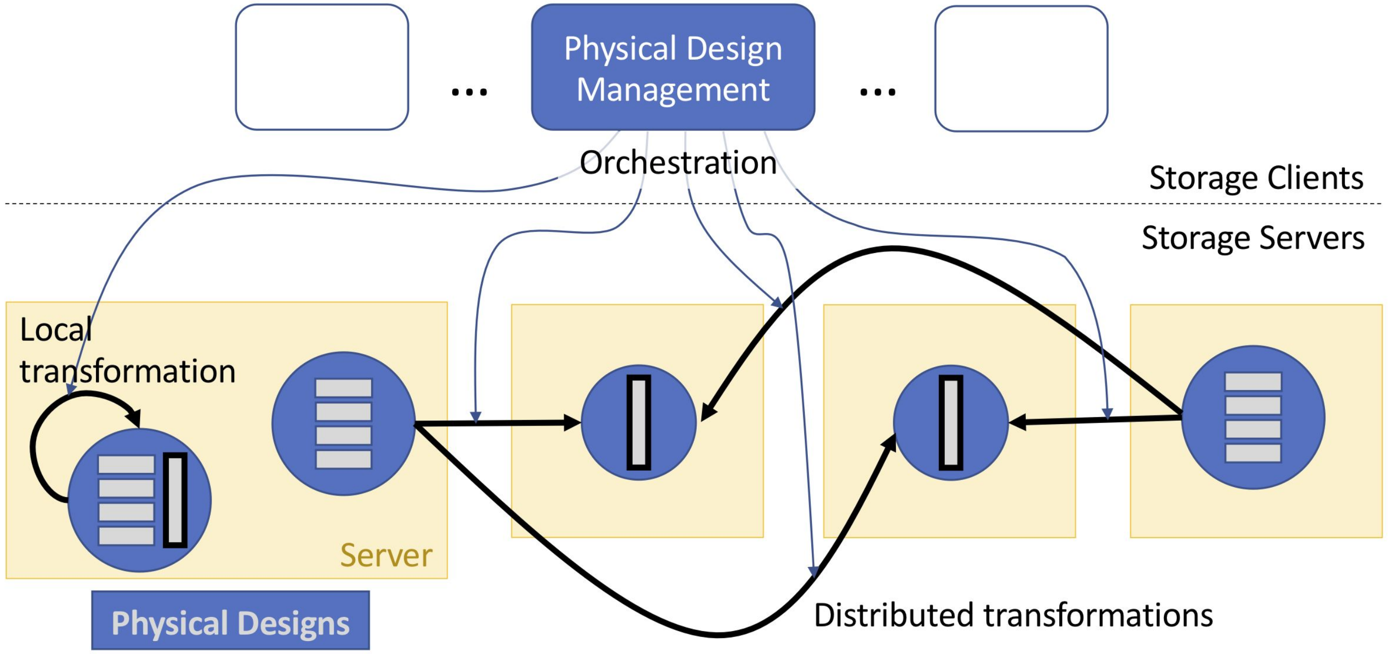
Physical Design Management

- Map a dataset to storage devices
  - Including metadata, views, indexes, data format, distribution
- Physical design management problem
  - *Identifying and executing a dataset transformation that will reduce workload processing cost without changing logical structure of dataset*
- Physical design has **topology** and **geometry**
  - Topology of logical structure is invariant under transformations
  - Geometry is particular mapping of topology to storage devices

Design management in practice

- Identify and execute transformations
  - Offline or online execution
- Execute transformations transparently to applications
  - Access is adapted to utilize a new design
  - Old designs deleted or retained for redundancy (row to col)
- May be executed in parallel
  - Independent of other transformations
  - DAG for dependent transformations

Physical Design Management Overview



Research Challenges

- Transformation execution
  - Managing resources for transform v. workload execution
  - How to navigate trade-off of transformation time v. workload performance?
- Transformation schedule
  - Slow roll-out with few resources scheduled initially?
  - Dedicate more resources earlier to realize benefits sooner?
- Managing metadata of transformation status so that workload can take advantage

Using Object Storage

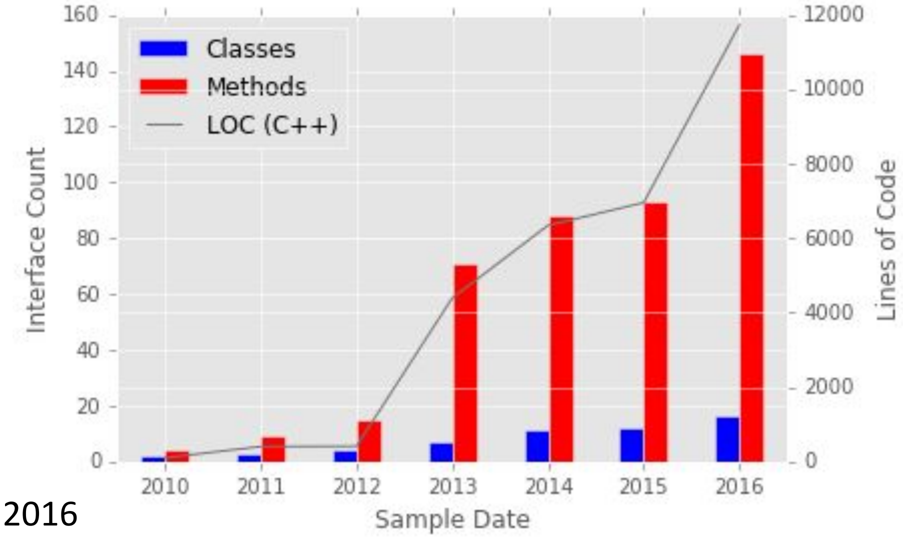
- Ceph distributed object storage
  - File, block, object interfaces
  - Embedded KV store on each server (indexing/metadata)
- Extensible object methods
  - Users can create custom

    object classes and methods
  - /src/cls/*
  - Cpp and lua interfaces
- Users already doing this

Sevilla et. al 2016

Using Object Storage

- Ceph distributed object storage
  - File, block, object interfaces
  - Embedded KV store on each server (indexing/metadata)
- Extensible object methods
  - Users can create custom object classes and methods
  - /src/cls/*
  - Cpp and lua interfaces
- Users already doing this



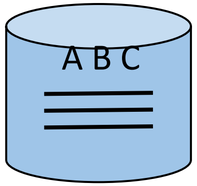
Sevilla et. al 2016

Serialization Libraries

- We embed usage of serialization libraries into our custom object classes within Ceph
  - Can use APIs to process data locally
- Google Flatbuffers (used for our row oriented layout)
  - Ordered contiguous sequence of bytes
  - Access individual elements without deserializing entire structure
  - Also use Flexbuffers
- Apache Arrow (used for our col oriented layout)
  - Optimized in-mem column-wise storage with compression

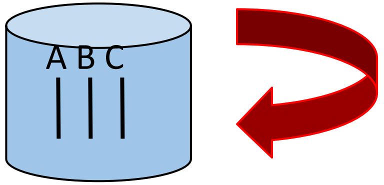
Local v. Distributed transformations

- **Local** transform is object-local
  - no network traffic



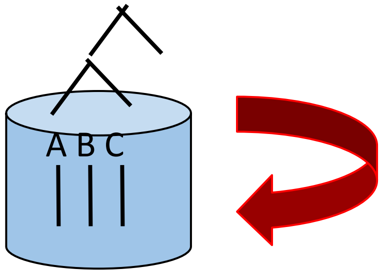
Local v. Distributed transformations

- **Local** transform is object-local
  - no network traffic
    - Primary - reorgs original data only



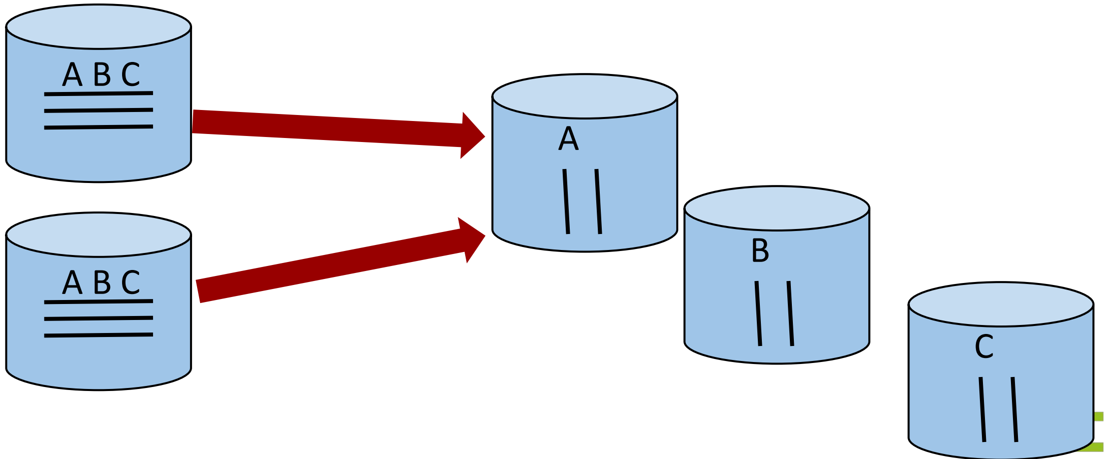
Local v. Distributed transformations

- **Local** transform is object-local
  - no network traffic
    - Primary - reorgs original data only
    - Secondary - creates auxiliary data (indexes, new metadata)

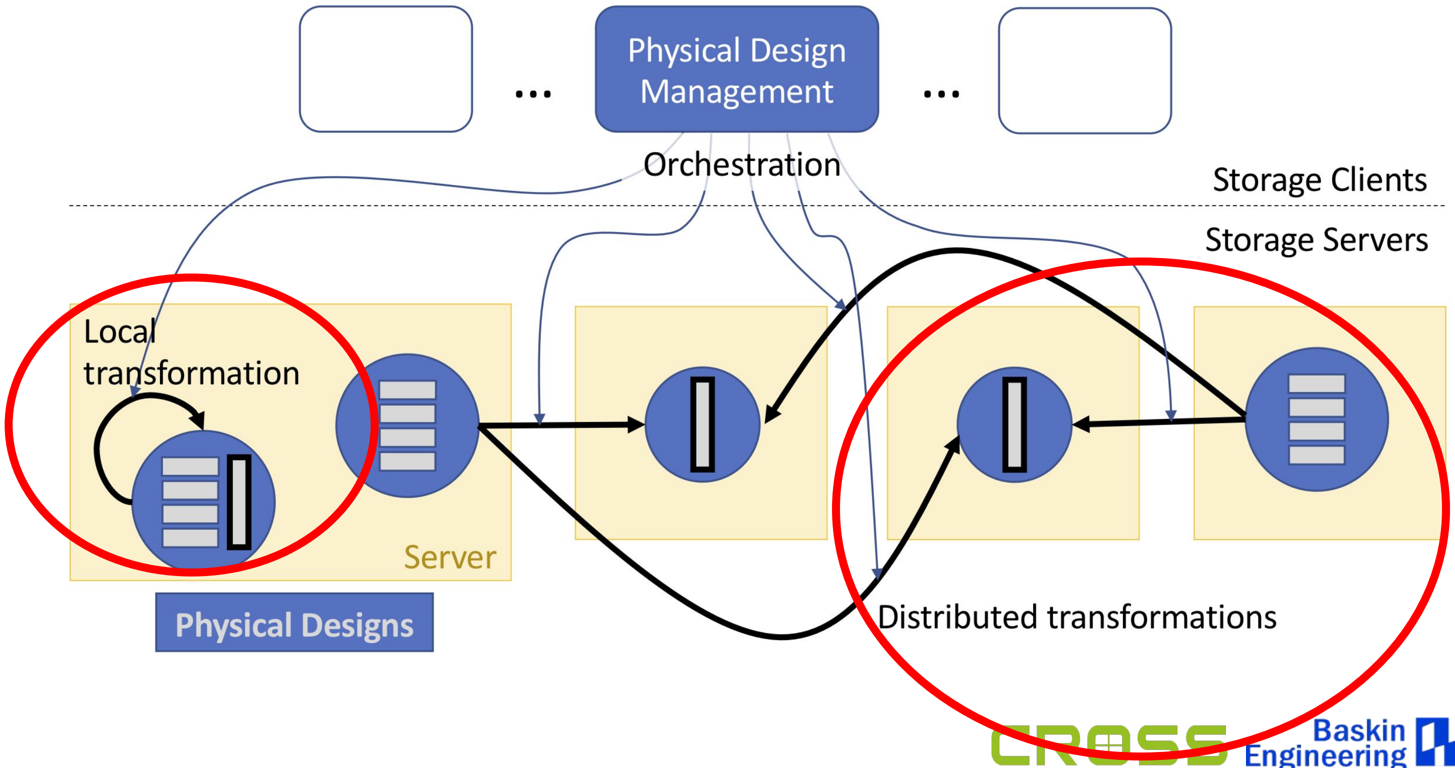


Local v. **Distributed** transformations

- **Distributed** transform is cross-object
  - Network traffic within storage layer only



Physical Design Management Overview



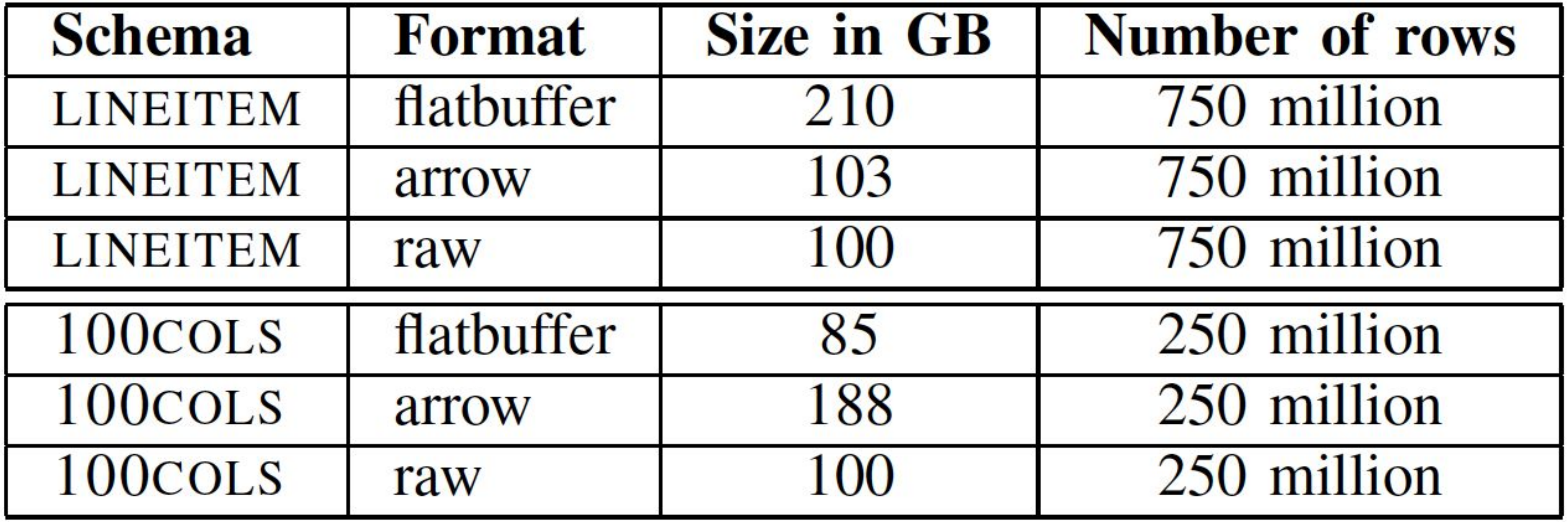
Implementation

- Created Ceph object classes (/src/scls/tabular)
  - Included Flatbuffers and Arrow Libraries in our source
  - Select, Project, Aggregate methods
- Local transformations use our object methods
- Distributed transformations use Ceph's existing **`copy_from()`** function (modified) along with our object methods

Evaluation

- Datasets and workloads
  - TPC-H lineitem table
  - IoT inspired data with 100 cols of integers
  - Select (1,10,100%), project, transform local and distributed
- Execution Environment
  - Cloudlab (c220g5 nodes) 40 cores, HDD, 10GbE
  - 1 client server
  - 4,8 storage servers (OSDs)
  - Ceph Luminous with our extensions (SkyhookDM project)
  - Avg of 3x execution, clearing FS cache each time

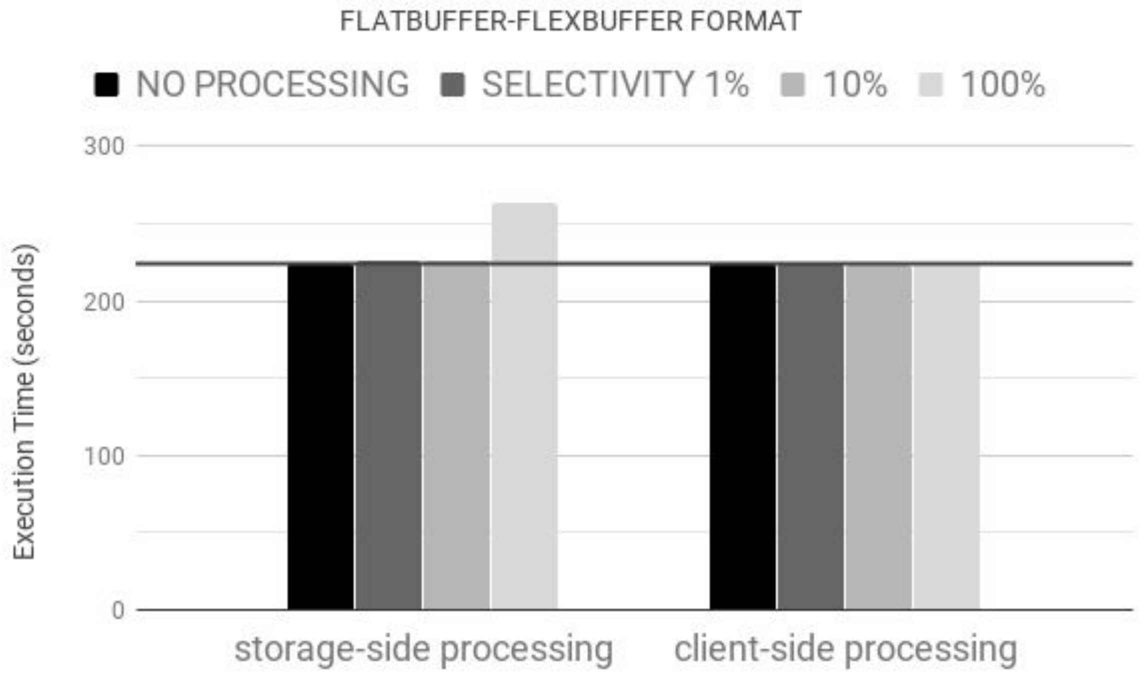
Dataset Sizes and Formats



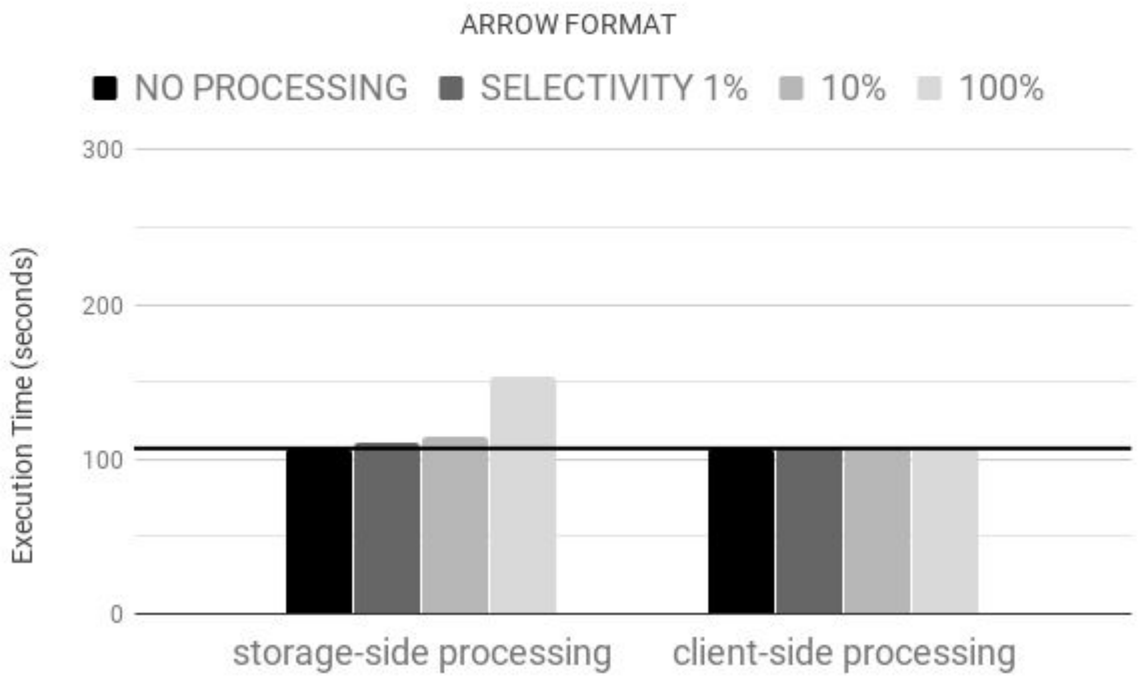
Schema	Format	Size in GB	Number of rows
LINEITEM	flatbuffer	210	750 million
LINEITEM	arrow	103	750 million
LINEITEM	raw	100	750 million
100COLS	flatbuffer	85	250 million
100COLS	arrow	188	250 million
100COLS	raw	100	250 million

We create 10,000 objects of uniform size, based on 100GB raw data

SELECT 1,10,100% (row format)



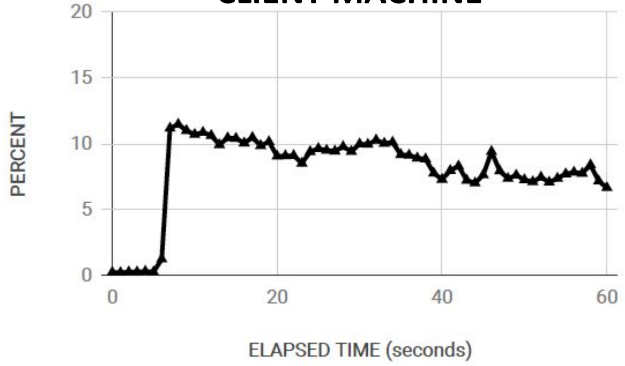
SELECT 1,10,100% (col format)



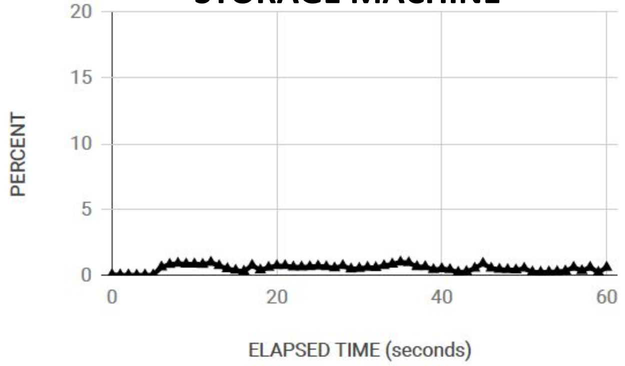
CPU Resource Usage (SELECT 1%)

CLIENT-SIDE PROCESSING (CPU)

CLIENT MACHINE



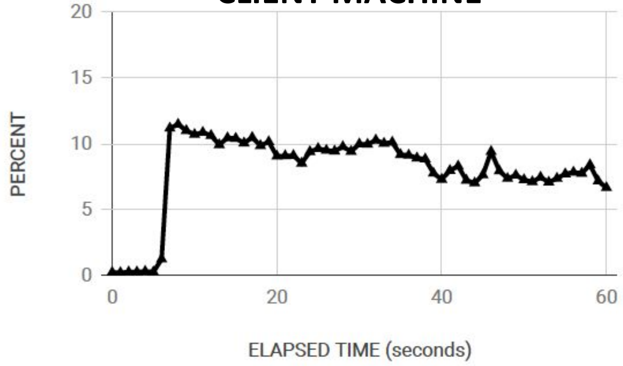
STORAGE MACHINE



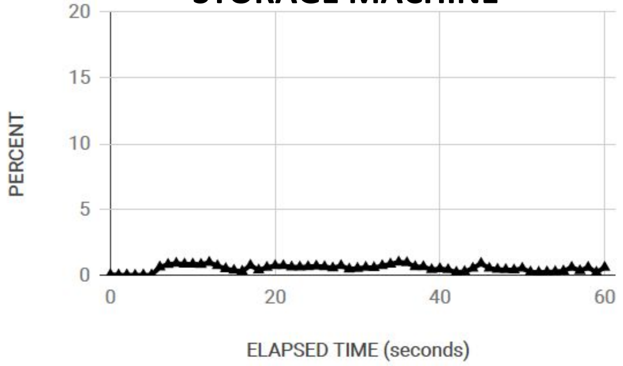
CPU Resource Usage (SELECT 1%)

CLIENT-SIDE PROCESSING (CPU)

CLIENT MACHINE

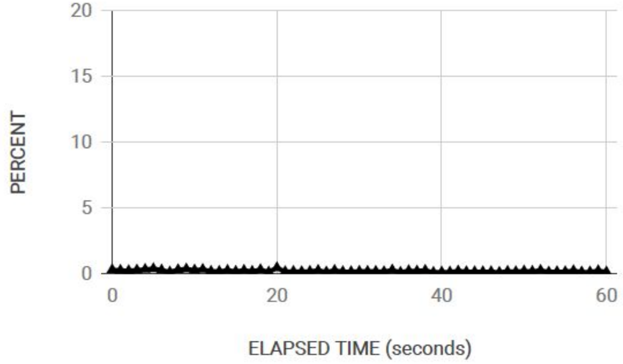


STORAGE MACHINE

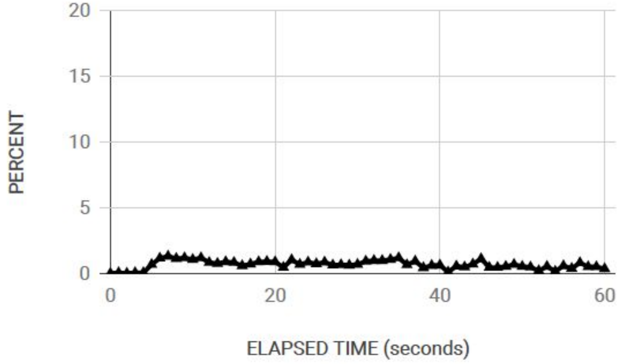


SERVER-SIDE PROCESSING (CPU)

CLIENT MACHINE



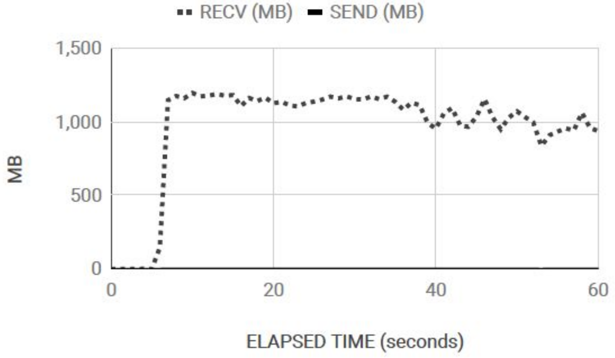
STORAGE MACHINE



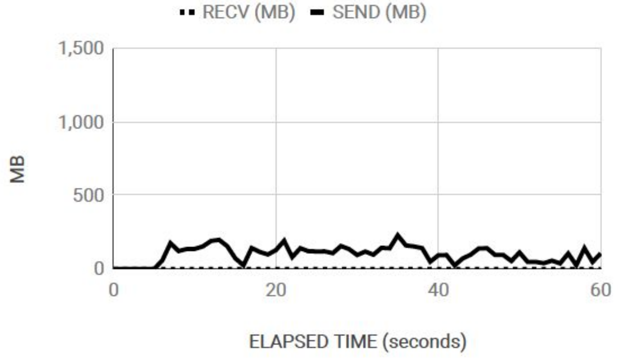
NET Resource Usage (SELECT 1%)

CLIENT-SIDE PROCESSING (NET)

CLIENT MACHINE



STORAGE MACHINE

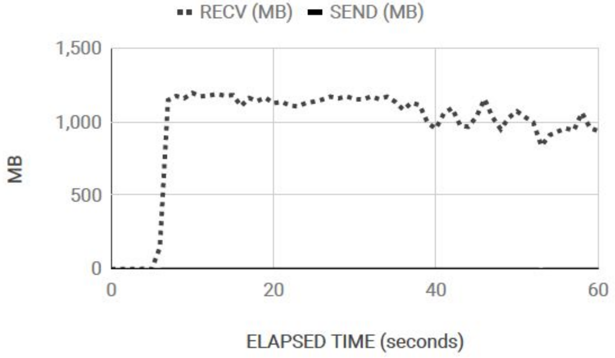


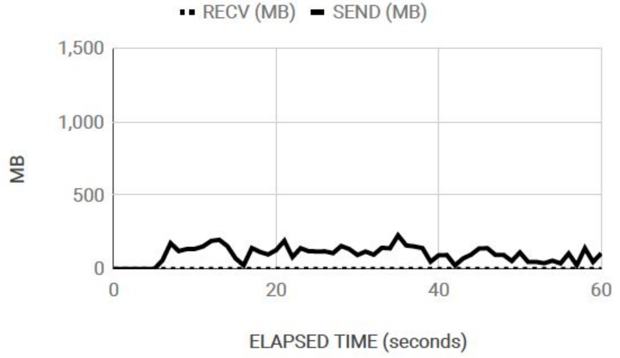
NET Resource Usage (SELECT 1%)

CLIENT MACHINE

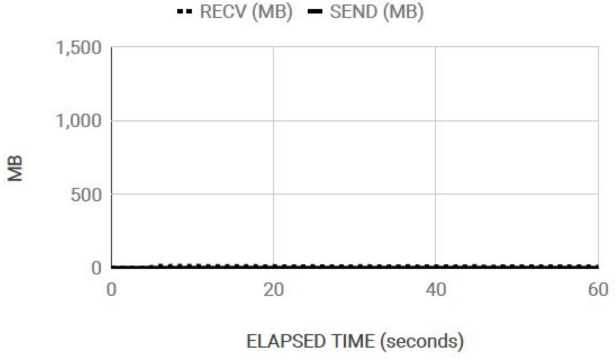
STORAGE MACHINE

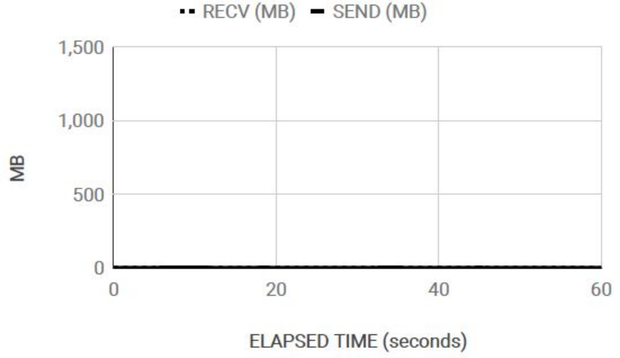
CLIENT-SIDE PROCESSING (NET)



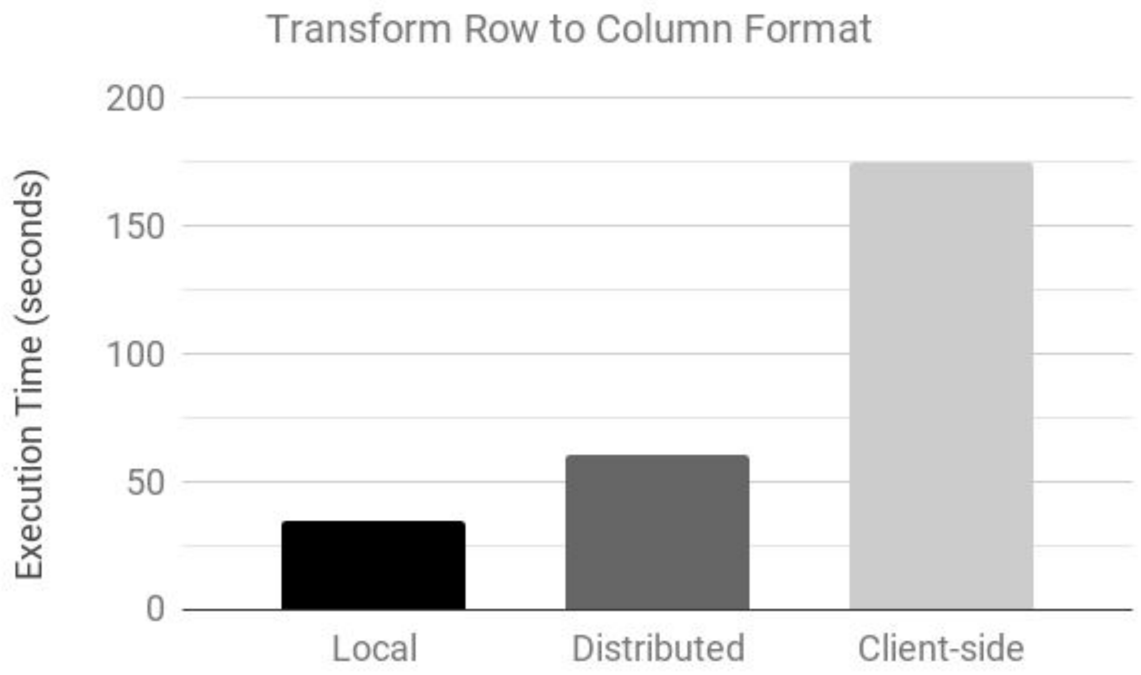


SERVER-SIDE PROCESSING (NET)

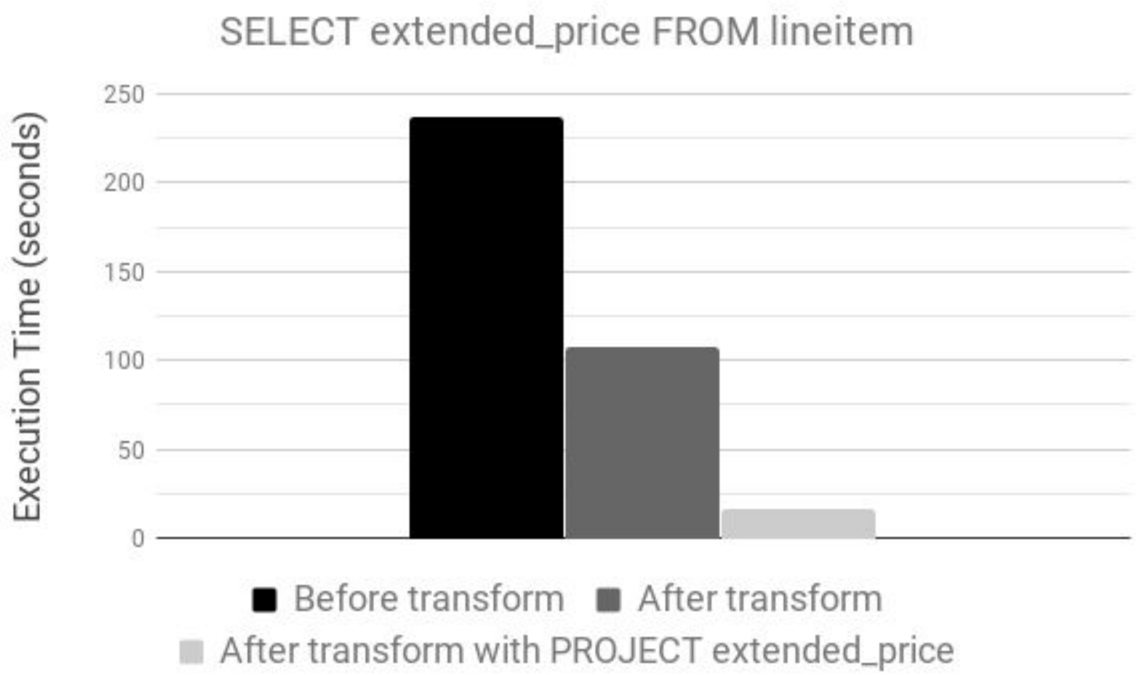




Transform time



Transform with PROJECT 1 column



Conclusion

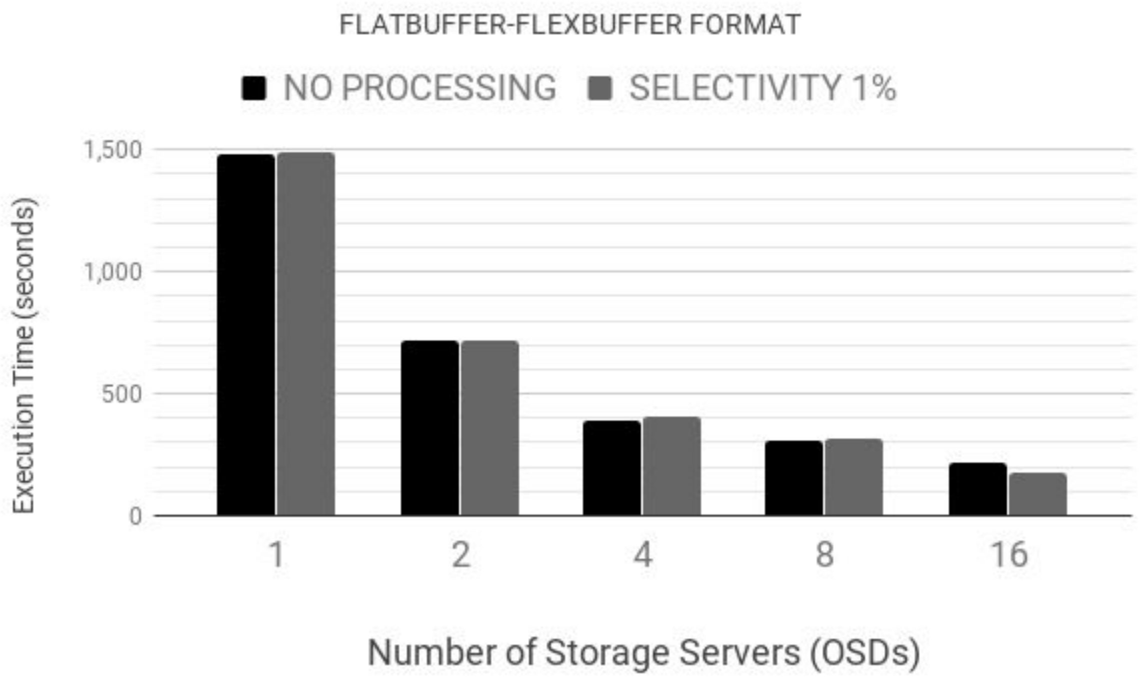
- Physical design management utilizes computational storage
  - but affects both the **type** of computations and their **performance** due to design reorganizations
- Implemented custom object classes in Ceph with fast serialization libraries and data semantics
  - Supports both processing and transformation
  - Objects can process and reorganize themselves
- Evaluated performance and resource usage before and after transformations
- Showed flexibility over different datasets, formats, selectivities

Thank you

- Acknowledgements
  - Center for Research in Open Source Software at UCSC
  - NSF Grant OAC-1836650, CNS-1764102, CNS-1705021

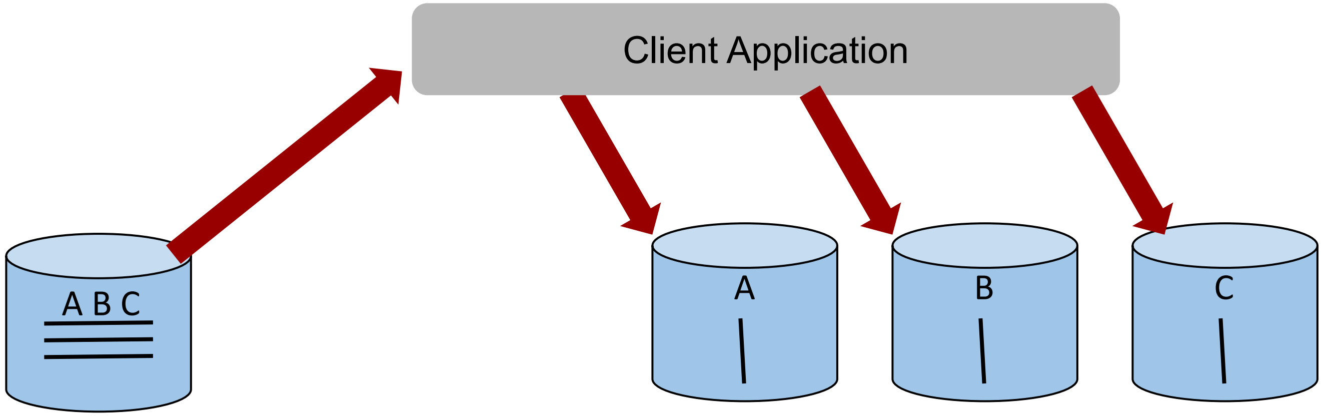
Backup slides

Scalability



v. **Client-side** transformations

- **Client** transform reads and writes data between client and storage


Client Application
A B C
A
B
C