Millisort:
An Experiment in Granular Computing

Seo Jin Park
with Yilong Li, Collin Lee and John Ousterhout

Platform Lab
Stanford University
Massively Parallel Granular Computing

- **Massively parallel computing as an application of granular computing**
  - Split longer time period jobs in finer grain to reduce end-to-end latency
  - Example) *ExCamera*: use 3600x threads to reduce video encoding time (1hr -> 1 sec)

- **Splitting jobs often hurts**
  - Many problems require much coordination among nodes
  - Coordination cost may limit the parallelism

- **Started a case study on distributed sorting problem**
  - Understand the coordination cost and the limit of parallelism
  - Build experiences on minimizing the cost of coordination
  - Serves as an application to drive building granular computing platform
Overview of Millisort

- **Question**: How much can we sort within 1-2 ms given unlimited nodes?

- **Millisort**: A new sorting algorithm tailored for massive parallelism
  - Minimizes overhead of coordinating 1000 nodes.
    - Simplifies coordination metadata that needs to be exchanged.
    - Leaves the metadata distributed in a smaller set of servers (e.g., 100 nodes).

- **Estimated result (no implementation yet)**
  - Using 1024 nodes, sorts 30 million tuples in 1.8 ms. (Single node sorting: 624 ms)
Distributed Sorting Problem

- Data are already evenly spread over 1000 nodes.
- After sorting, the sorted data (almost) evenly distributed to 1000 nodes.
- 10B keys, 90B value
- Given time: 1-2 ms, which is not very long!
  - A node can sort ~70,000 tuples (optimized radix sort with 12 cores)
  - Can invoke 200 RPCs sequentially (each RPC takes 5-10 µs)
  - On 40G network, can send 4MB (~40,000 tuples)
  - 10,000 sequential cache misses
Overview of Millisort

A variant of the standard distributed bucket sorting

1. **Local sort**
   Each node sorts its partition

2. **Partitioning**
   Figure out boundaries of sorted buckets

3. **Shuffling**
   Transfer data in each buckets to destinations

4. **Merging**
   Receive locally sorted tuples and merge them as they arrive
Stage 1. Local Sort

- Each node sort its partition internally
  - Speeds up later stages significantly
- Use existing radix sort
  - With 12 cores, each node can sort 15 ns / item
  - With 1000 nodes (12 cores on each),
    - For 1 ms, we can sort about 67 million tuples (67k per node)
Stage 2. Partitioning

- Figure out bucket boundaries
- Traditional algorithms take too long!
  - Commonly used algorithm: histogram sampling & splitting
  - Estimated partitioning time for 1000 nodes: 500 ms (21 ms for 256 nodes)
- Millisort uses a totally new approach.
  - Guaranteed bound approximate algorithm
    - Largest bucket is twice as large as the perfect distribution
    - For even more distribution, we may use more pivots
  - Estimated partitioning time for 1000 nodes: ~400 µs (118 µs for 256 nodes)
For $M$ nodes cluster, each partition picks $M$ equally spaced pivots

- Sort the $M^2$ pivots all together.
  - This is a non-trivial problem
  - Strawman: coalesce all $M^2$ pivots to a machine => too slow.
  - Solution: use the same distributed bucket sorting using fewer nodes.

- Pick $M^{th}$, $2M^{th}$, $3M^{th}$, …, $M^2$ th pivots. They are the $b_1$, $b_2$, $b_3$, …, $b_{1000}$

- Broadcast the bucket boundaries $< b_1, b_2, b_3, …, b_{1000}>$

Network bandwidth on the machine
- 10B key * 1 million pivots = 10MB
- 10MB / 40 Gbit/s => 2 ms
Stage 2.b Sorting $M^2$ pivots

Sorted $M (= 1000)$ pivots

Coalesce pivots to $M/10$ nodes

Pick $M/10 (= 100)$ super-pivots

Coalesce all $\frac{M^2}{100}$ super-pivots to a node & select super bucket boundaries

Shuffle all pivots & merge sort

Broadcast bucket boundaries

Estimated Time

- 86 µs
- 71 µs
- 20 µs
- 10 µs
- 10 µs
- 172 µs
- 30 µs

Total: 399 µs
1. **Local sort**
   Each node sorts its partition

2. **Partitioning**
   Figure out boundaries of sorted buckets

3. **Shuffling**
   Transfer data to destinations

4. **Merging**
   Receive locally sorted tuples and merge them as they arrive

---

**Estimated Time**

~1 ms
Expected Performance

- **Built performance estimator for Millisort** (real implementation is in progress)
- **Anticipated cluster configurations**
  - > 1000 nodes (10 cores / node)
  - 40 Gbits/s full-bisection networking
  - 30 GB/sec peak memory bandwidth
  - 0.3 µs to send message, 0.7 µs to receive message
- **Sorting problem**
  - 50 million tuples are evenly distributed to 1000 machines
  - Each tuple: 10B key, 90B value ➔ Total data size: 5 GB (5 MB per node)
- **Estimated time to sort**
  - Single machine radix sort: ~1 sec
  - Millisort on 1000 machines: 2.7 ms (1 ms of local radix sort + 0.5 ms of partitioning + 1.2 ms of shuffling)

\[ \text{fundamental cost of sorting} + \text{fundamental cost of distribution} \]
Estimated Sort Time by Machine Count

- 50 million tuples, 10B key, 90B value
How many can you sort with infinite resources?

- Infinite # of nodes available
Conclusion

- **Sorting is a hard problem to use massively distributed computing**

- **Millisort hints the cost of distribution in granular computing**
  - Building experiences on minimizing the coordination cost
  - A specific case to understand coordination cost and the limit of parallelism
  - Reusable components
    - Multicast-style communications (broadcast, gather/merge, all-gather)
    - Performance simulator

- **Still at an early stage**
  - What we have: algorithm, cost model
  - Hope to finish implementation this summer to verify

- **Hope to see**
  - System challenges to be solved to execute operations in tight latency budgets
  - New platform/programming model to run/develop Millisort efficiently