Distributed Aggregation for Data-Parallel Computing: Interfaces and Implementations

Yuan Yu, Pradeep Gunda, Michael Isard

presented by
Risi Thonangi

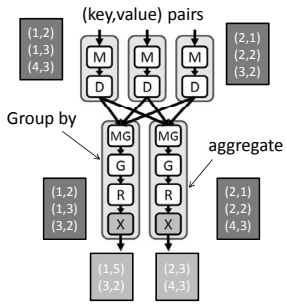
Topic of discussion

- Groupby-Aggregation

```
SELECT avg(Income) … GROUP BY Zip
```

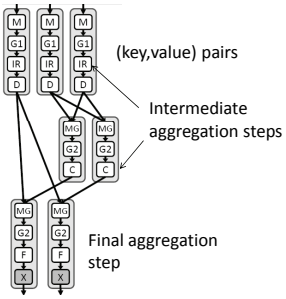
  - Aggregation with user-defined functions
- Compare various distributed executing systems for Groupby-Aggregation
  - MapReduce, Parallel DB, Dryad

Distributed Aggregation (example)



- Top layer distributes input
- Below layer aggregates
- Disadvantage
  - Lot of data in network

Partial aggregation for improved performance



- Save on
  - Network traffic
- Distribute computation
- Costs?
  - more nodes in the pipeline
  - Slightly more computation

Decomposable Functions

- Not all aggregation functions are decomposable
  - Median, …
- Function H is decomposable if
  - $H = C(I(x_1+x_2)) = C(I(x_1) + I(x_2))$
  - C and I are commutative
- H is associative-decomposable if
  - H is decomposable and
  - $C(C(x_1+x_2)+C(x_3)) = C(C(x_1) + C(x_2+x_3))$

Next..

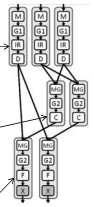
- Programming interfaces
  - Hadoop, Parallel DB, DryadLINQ
- Applications which need Groupby-Aggregation
- Implementations for distributed aggregation
- Experimental evaluation

Hadoop's interface

```
// InitialReduce: input is a sequence of raw data tuples;
// produces a single intermediate result as output
static public class Initial extends EvalFunc<Tuple> {
  @Override public void exec(Tuple input, Tuple output)
      throws IOException {
    try {
      output.appendField(new DataAtom(sum(input)));
      output.appendField(new DataAtom(count(input)));
```

```
// Combiner: input is a sequence of intermediate results;
// produces a single (coalesced) intermediate result
static public class Interned extends EvalFunc<Tuple> {
  @Override public void exec(Tuple input, Tuple output)
      throws IOException {
    combine(input.getBagField(0), output);
  }
}
```

```
// FinalReduce: input is one or more intermediate results;
// produces final output of aggregation function
static public class Final extends EvalFunc<DataAtom> {
  @Override public void exec(Tuple input, DataAtom output)
      throws IOException {
    Tuple combined = new Tuple();
    if(input.getField(0) instanceof DataBag) {
      combine(input.getBagField(0), combined);
    } else {
      throw new RuntimeException([...]);
    }
    double sum = combined.getAtomField(0).numval();
    double count = combined.getAtomField(1).numval();
```



- Hadoop's is an Iterator interface
- Comments
  - Programmer has to marshal data to/fro system types
  - Interface is restricted

Oracle's interface

```
MEMBER FUNCTION ODCIAggregateIterate
  ( self IN OUT AvgInterval,
    val IN DSINTERVAL_UNCONSTRAINED
  ) RETURN NUMBER IS
BEGIN
  self.runningSum := self.runningSum + val;
  self.runningCount := self.runningCount + 1;
  RETURN ODCIConst.Success;
```

Accumulator-based interface

Comments

- Not much difference between iterator and accumulator-based interfaces
- Weak point
  - Complex aggregation functions and data types are difficult to express

DryadLINQ

- Dryad + LINQ framework
- Allows tight integration between programming language and execution engine
  - No overhead due to type casts/conversion
- Use a native programming language (as in Hadoop)
- Dryad implements both iterator and accumulator-based interfaces

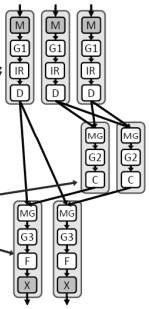
Accumulator Interface in DryadLINQ

```
[Decomposable("Initialize", "Iterate", "Merge")]
public static IntPair SumAndCount(IEnumerable<int> g) {
  return new IntPair(g.Sum(), g.Count());
}

public static IntPair Initialize() {
  return new IntPair(0, 0);
}

public static IntPair Iterate(IntPair x, int r) {
  x.first += r;
  x.second += 1;
  return x;
}

public static IntPair Merge(IntPair x, IntPair o) {
  x.first += o.first;
  x.second += o.second;
  return x;
}
```

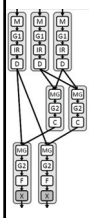


Slide from Yuan Yu, SOSP09

Decomposable aggregation functions

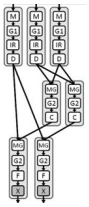
- Aggregation function is decomposable if every leaf in its expression tree is
  - A constant
  - g.key
  - Or H(g) for a decomposable function H
- Examples
  - Average, standard deviation, …

DryadLINQ Implementations for Groupby-Aggregate



FullSort	PartialSort	Accumulator-FullHash	Accumulator-PartialHash
•sort all input •IR for unique key •distribute	•sort each chunk •IR for unique key •distribute	•Hash input •distribute	•Partial Hash •distribute
•Merge inputs •Combine and forward	•Full sort of inputs •Combine and forward	•Hash input •Combine and forward	•Full Hash •Combine and forward
•Merge inputs •Final reduce	•Merge inputs •Final reduce	•Hash •Final reduce	•Hash •Final reduce
-main memory at level-1 - heavy sort in level-1 + max. reduction + pipe-lined exec	-More network traffic + smaller sorts + pipe-lined exec from level-2	- No pipe-lining (?) + Less memory + no sorting + max. reduction	- Full Hash in level-2 + Less memory + no sorting + good reduction

DryadLINQ Implementations for Groupby-Aggregate (contd.)



Iterator FullHash

- •Accumulate input in hash buckets •Reduce & distribute
- •Iterator FullHash •Combine and forward
- •Hash •Final reduce

- Memory at level-1
- Full Hash at level-2
- No pipelining
+ no sorting
+ good reduction

Iterator PartialHash

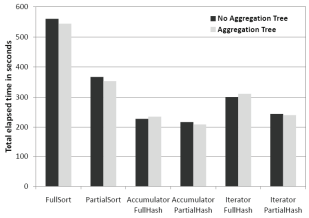
- •Similar to Accumulator-PartialHash •Reduce & distribute
- •Iterator-FullHash •Combine and forward
- •Hash •Final reduce

- Memory at level-1
- Full Hash at level-2
- No pipelining
+ no sorting
+ Partial reduction

Experimental Evaluation

- 3 aggregation functions
  - Word statistics
  - Word top documents
  - PageRank
- Execution Environment
  - 236 computers
  - Two-level network

Word statistics (& popularity)



- Intermediate aggregation is not helping
  - Extra node in the pipeline
  - Extra Disk I/O & node initialization costs
- Sort implementations are slow
  - sort complexity
- Acc-FullHash is worse than Acc-PartialHash
  - May be because of the small partial-hash table

Reduction strategy	No Aggregation	Aggregation
FullSort	[11.7, 4.5]	[11.7, 2.5, 1.8]
PartialSort	[3.7, 13.7]	[3.7, 7.3, 1.8]
Acc-FullHash	[11.7, 4.5]	[11.7, 2.5, 1.8]
Acc-PartialHash	[4.6, 11.4]	[4.6, 6.15, 1.85]
Iter-FullHash	[11.7, 4.5]	[11.7, 2.5, 1.8]
Iter-PartialHash	[4.1, 12.8]	[4.1, 6.6, 1.9]

Summary

- DryadLINQ has much richer interface than Hadoop
- DryadLINQ has better integration with programming languages than parallel DBs
- 6 plans for groupby-Aggregate in DryadLINQ

Questions?