Design Pattern for Scientific Applications in DryadLINQ CTP

DataCloud-SC11

Hui Li
Yang Ruan, Yuduo Zhou
Judy Qiu, Geoffrey Fox

PERVASIVE TECHNOLOGY INSTITUTE
INDIANA UNIVERSITY

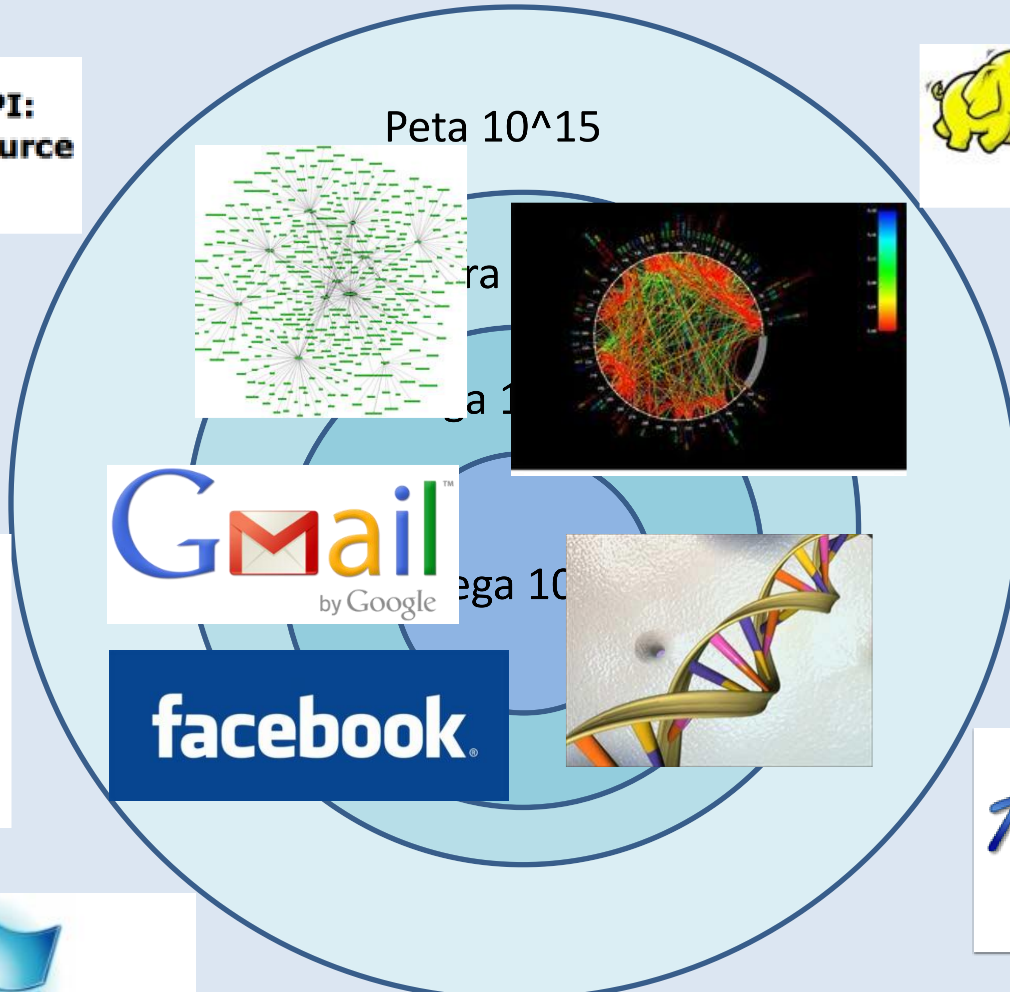
SALSA HPC

Motivation

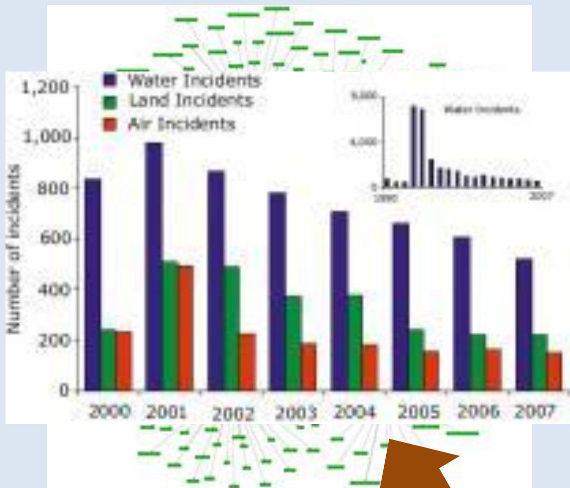
- One of latest release of DryadLINQ was published on Dec 2010
- Investigate usability and performance of using DryadLINQ language to develop data intensive applications
- Generalize programming patterns for scientific applications in DryadLINQ

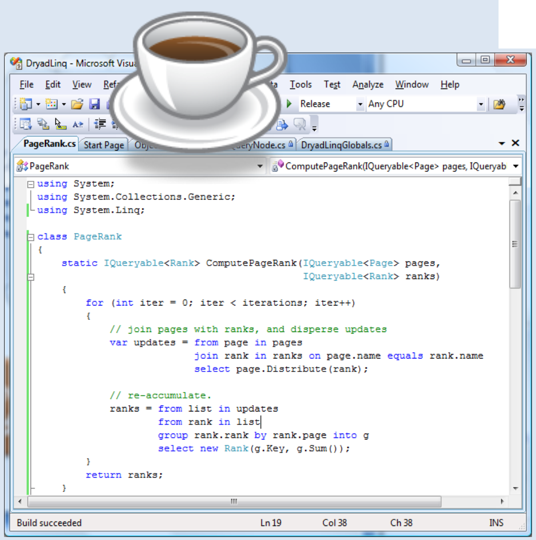
Big Data Challenge

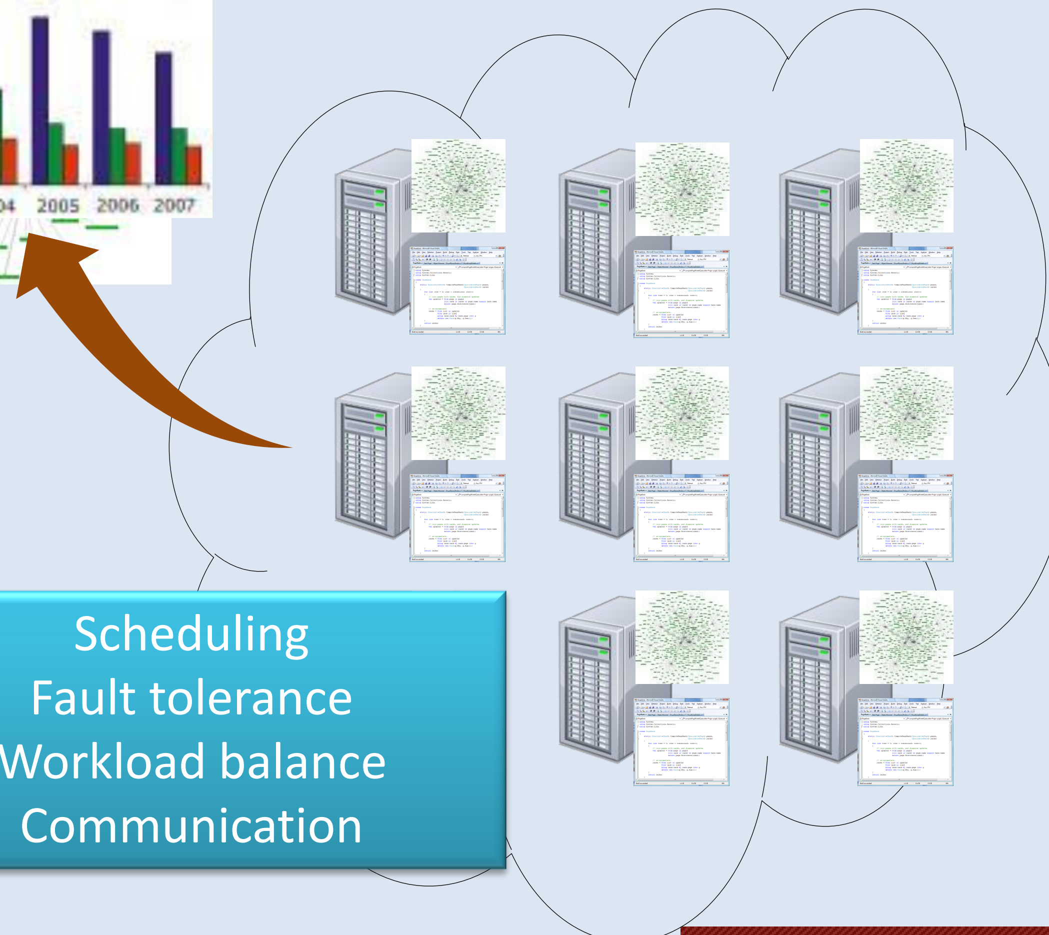
Open MPI: Open Source HPC



MapReduce Processing Model







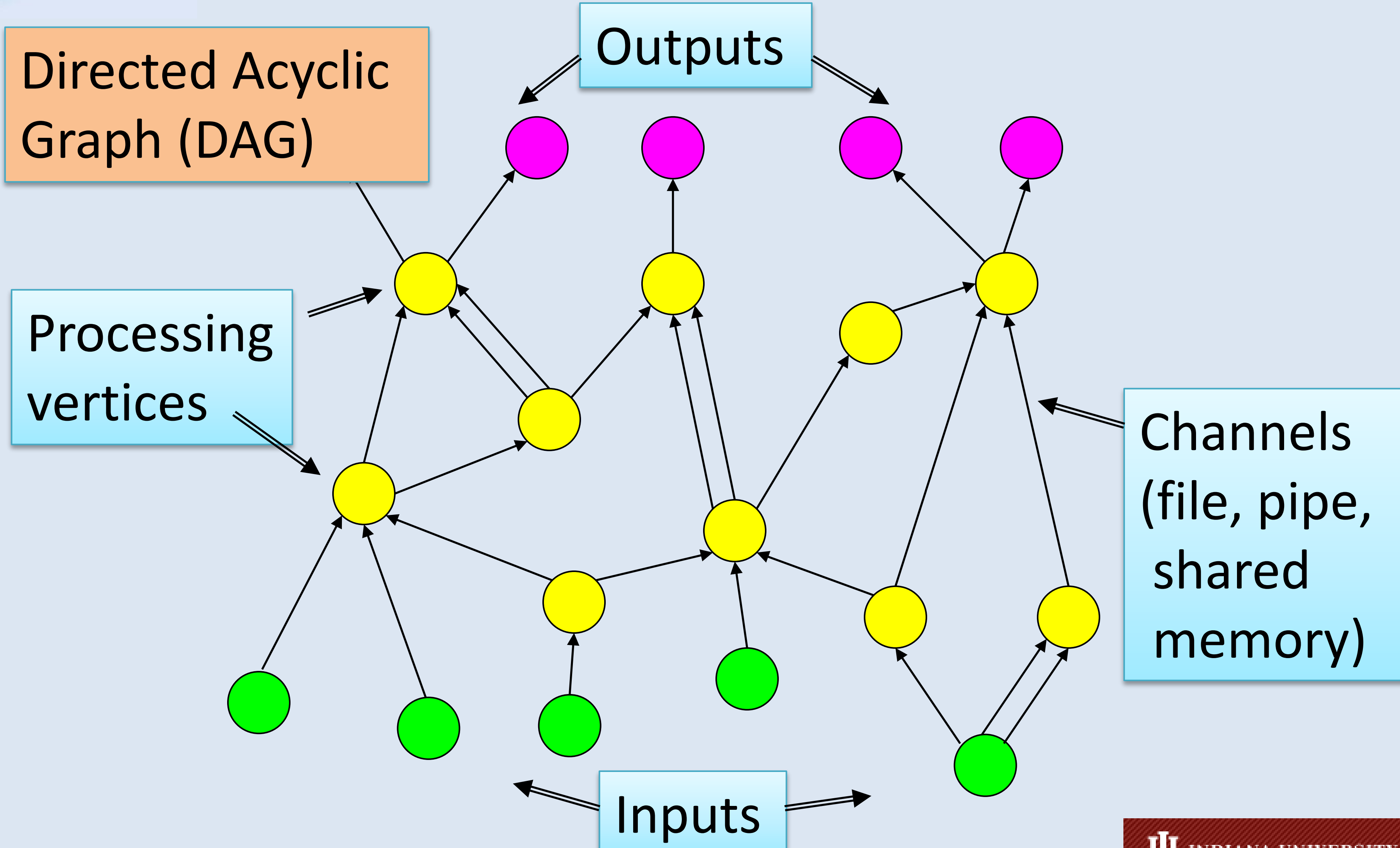
Scheduling
Fault tolerance
Workload balance
Communication

INDIANA UNIVERSITY

Disadvantages in MapReduce

- Rigid and flat data processing paradigm are difficult to express semantic of relational operation, such as Join.
- Pig or Hive can solve above issue to some extent but has several limitations:
  - Relational operations are converted into a set of MapReduce tasks for execution
  - For some equal join, it needs to materialize entire cross product to the disk
- Sample: MapReduce PageRank

Microsoft Dryad Processing Model



Microsoft DryadLINQ Programming Model

- Higher level programing interface for Dryad
- Based on LINQ model
- Unified data model
- Integrated into .NET language
- Strong typed .NET objects

Common LINQ providers

Provider	**Base class**
<T>	`Strong typed .NET objects`
LINQ-to-objects	`IEnumerable<T>`
PLINQ	`ParallelQuery<T>`
DryadLINQ	`DistributedQuery<T>`

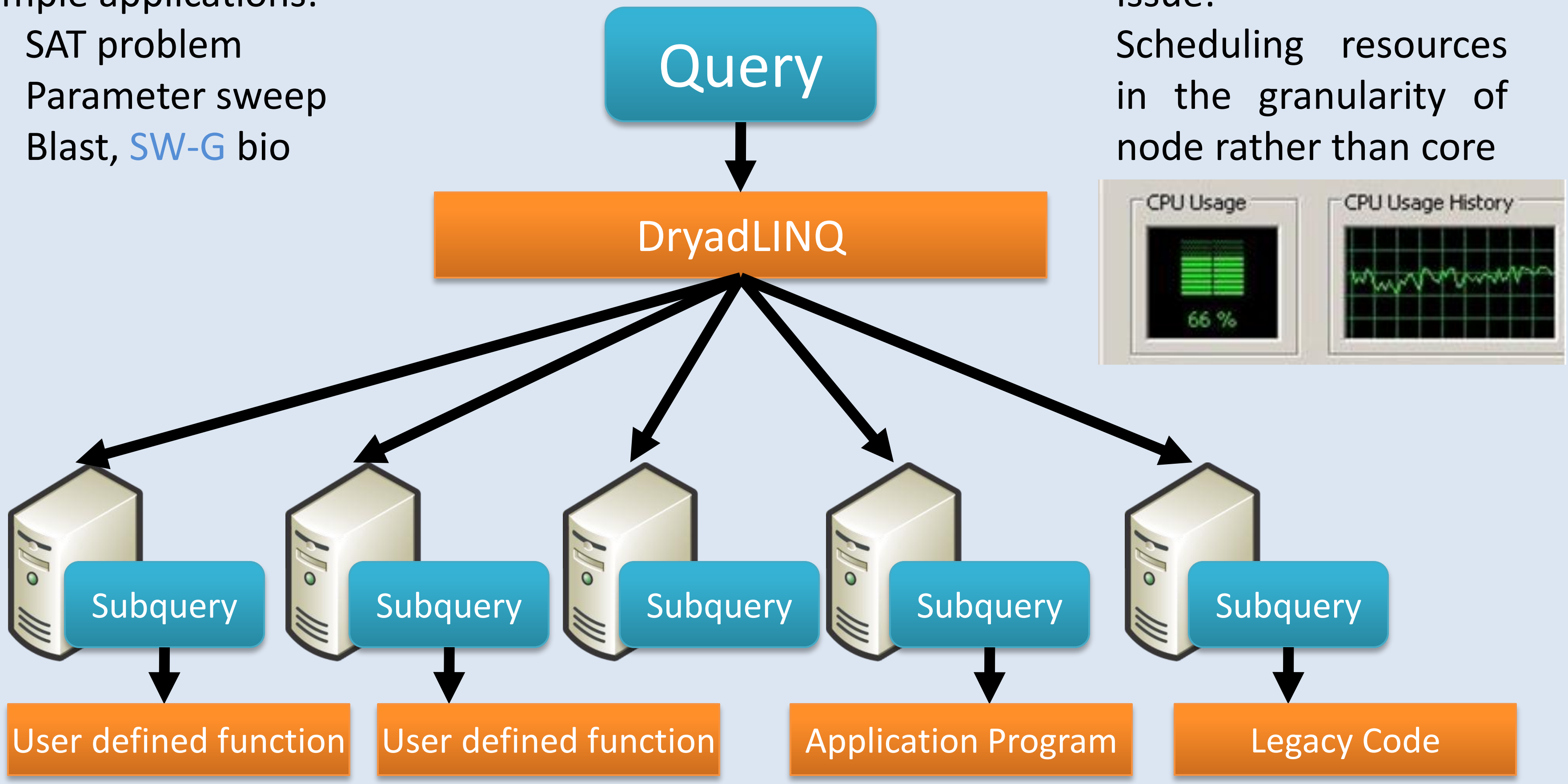
Pleasingly Parallel Programing Patterns

Sample applications:

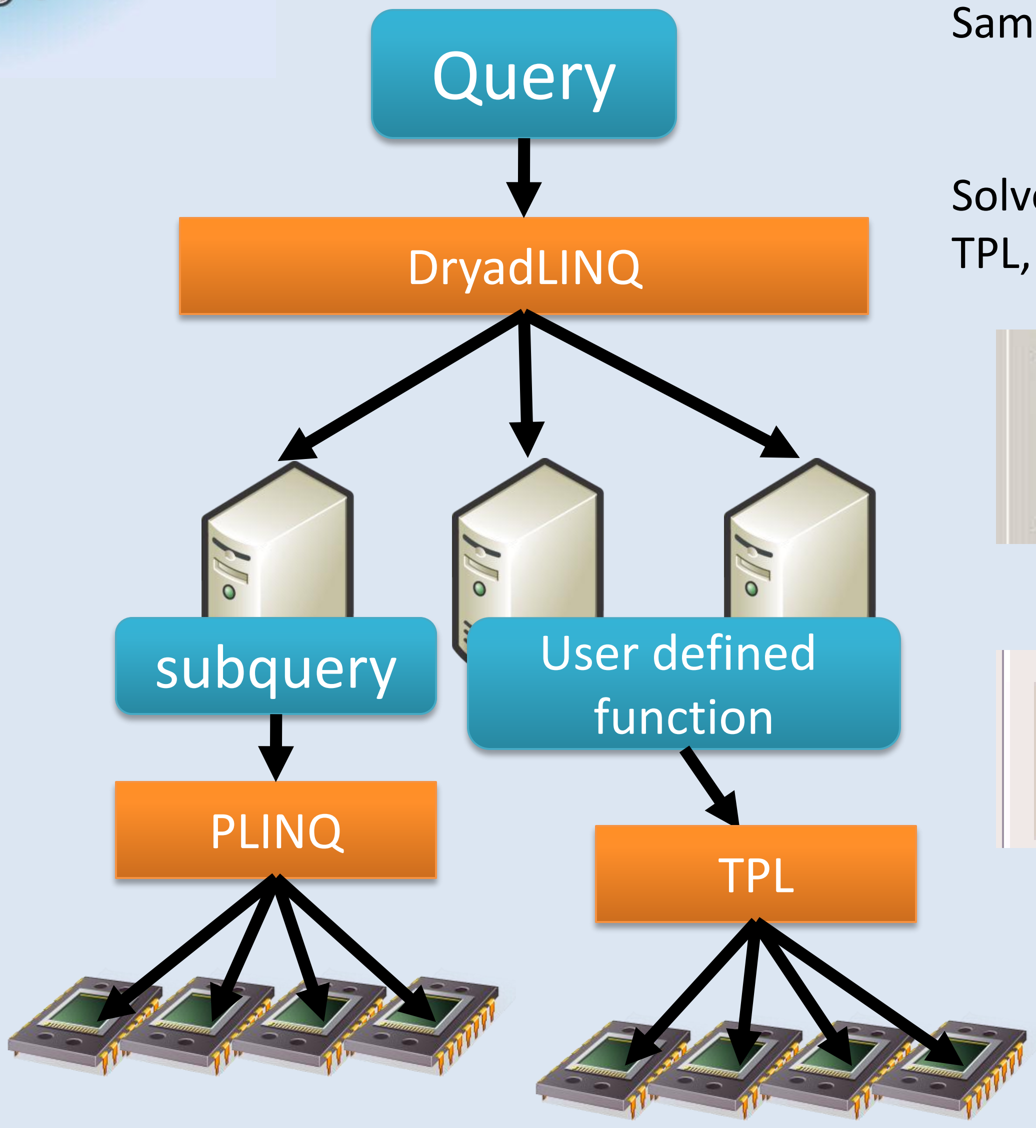
1. SAT problem
2. Parameter sweep
3. Blast, SW-G bio

Issue:

Scheduling resources in the granularity of node rather than core



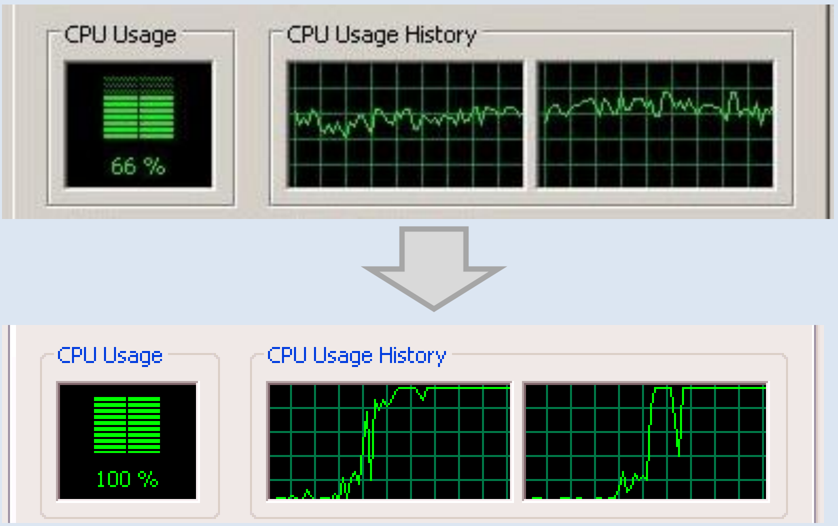
Hybrid Parallel Programming Pattern



Sample applications:

1. Matrix Multiplication
2. GTM and MDS

Solve previous issue by using PLINQ, TPL, Thread Pool technologies



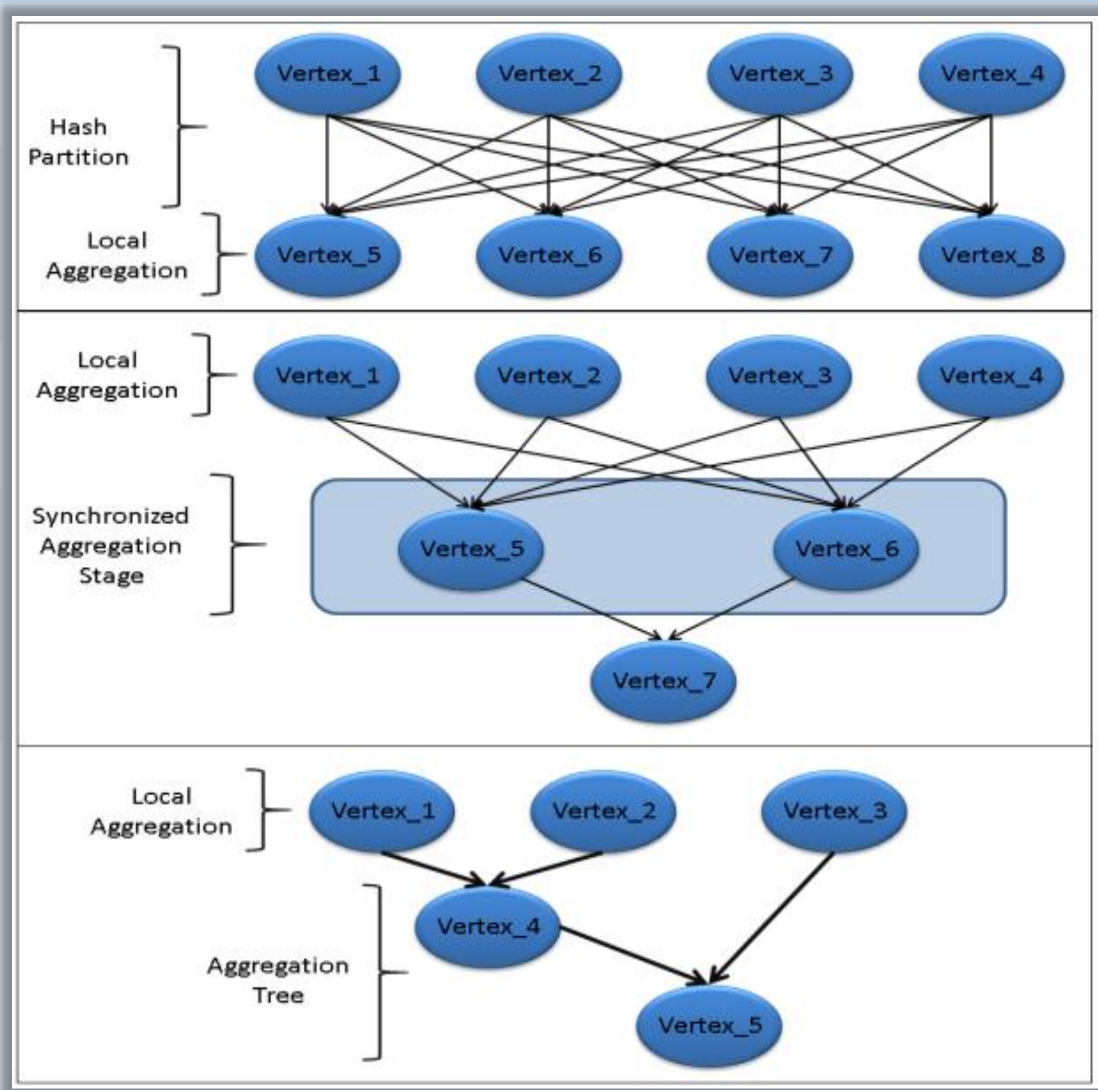
Distributed Grouped Aggregation Pattern

Sample applications:

1. Word count
2. PageRank

Three aggregation approaches:

1. Naïve aggregation
2. Hierarchical aggregation
3. Aggregation tree



INDIANA UNIVERSITY

Implementation and Performance

Hardware configuration

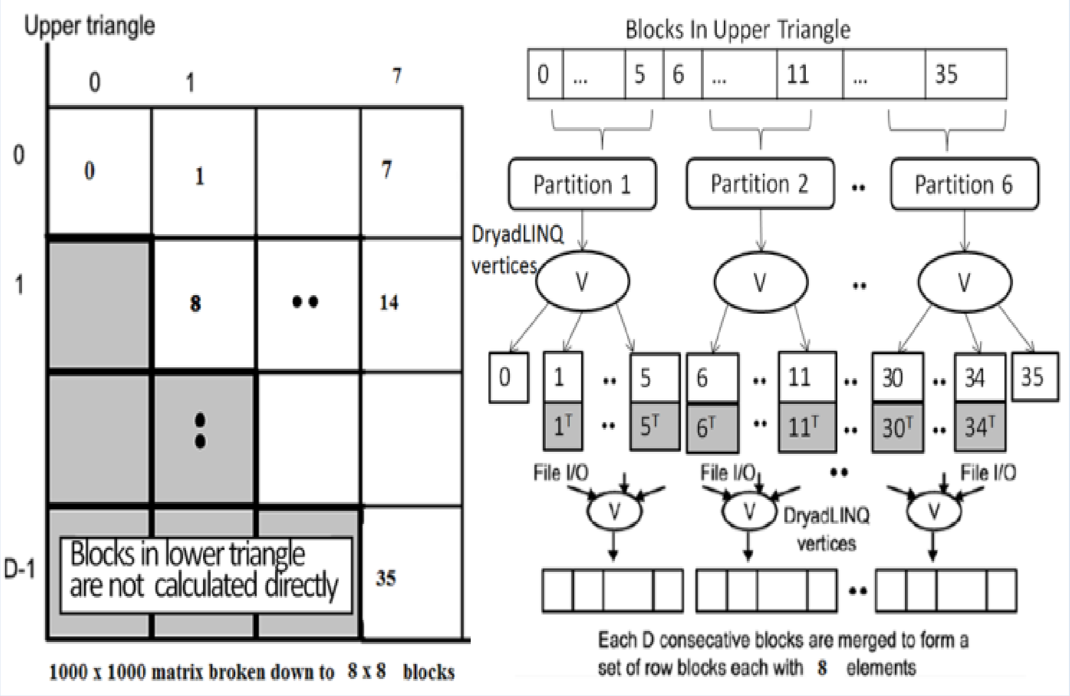
TEMPEST	TEMPEST	TEMPEST-CNXX
CPU	Intel E7450	Intel E7450
Cores	24	24
Memory	24.0 GB	50.0 GB
Memory/Core	1 GB	2 GB

STORM	STORM-CN01,CN02, CN03	STORM-CN04,CN05	STORM-CN06,CN07
CPU	AMD 2356	AMD 8356	Intel E7450
Cores	8	16	24
Memory	16 GB	16 GB	48 GB
Memory/Core	2 GB	1 GB	2 GB

1. We use DryadLINQ CTP version released in December 2010
2. Windows HPC R2 SP2
3. .NET 4.0, Visual Studio 2010

INDIANA UNIVERSITY

Pairwise sequence comparison using Smith Waterman Gotoh



1. Pleasingly parallel application
2. Easy to program with DryadLINQ
3. Easy to tune task granularity with DryadLINQ API

```
Var SWG_Blocks = create_SWG_Blocks(AluGeneFile, numOfBlocks, BlockSize)
Var inputs= SWG_Blocks.AsDistributedFromPartitions();
Var outputs= inputs.Select(distributedObject => SWG_Program(distributedObject));
```

Workload balance in SW-G

1. SWG tasks are heterogeneous in CPU time.
2. Coarse task granularity brings workload balance issue
3. Finer task granularity has more scheduling overhead

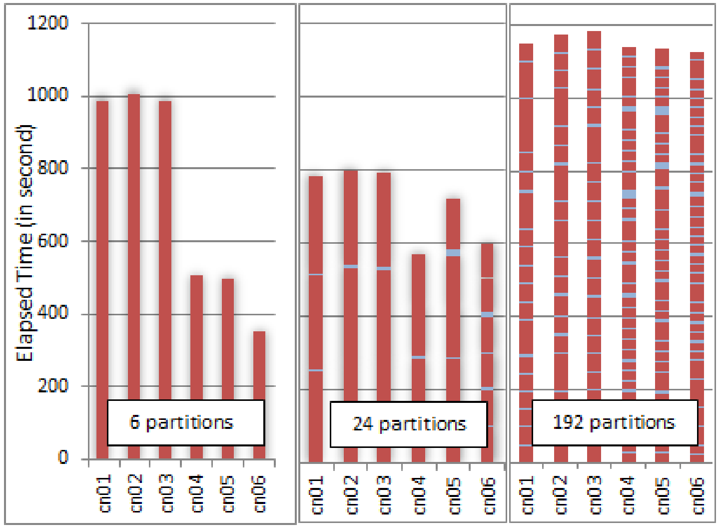
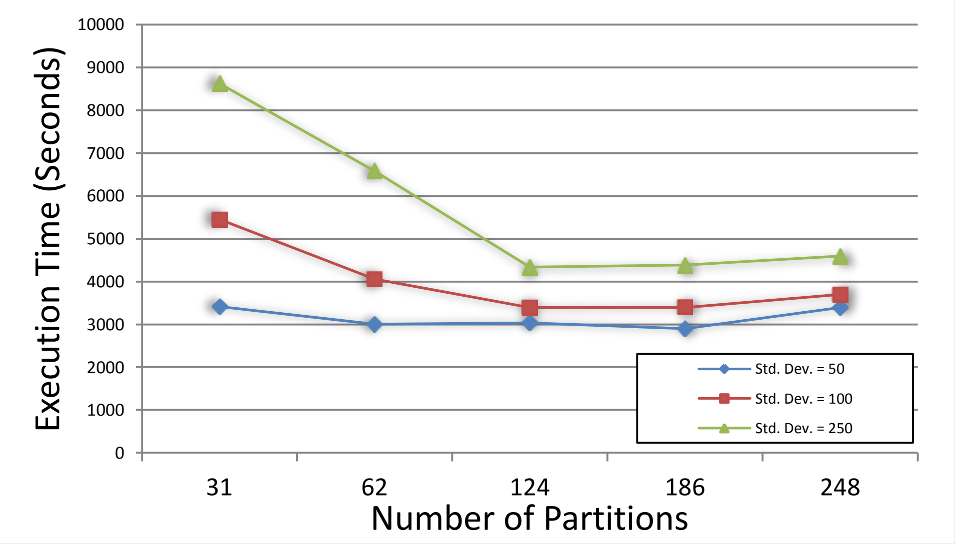


Fig. 5: CPU and Scheduling Time of the Same SW-G Job with Various Partition Granularities



Simply solution: tune the task granularity with DryadLINQ API

Related API:

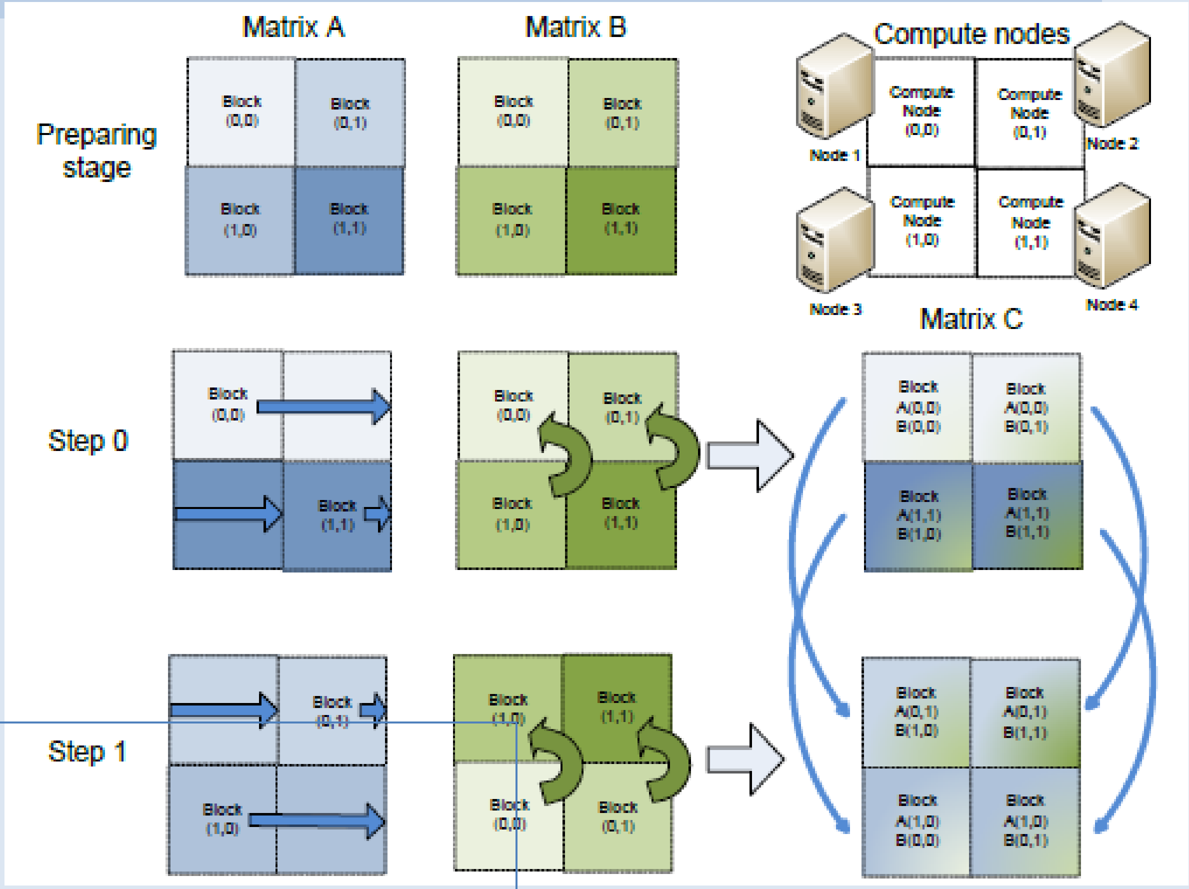
AsDistributedFromPartition()

RangePartition<Type t>()

Square dense matrix multiplication

Parallel MM algorithms:
1.Row partition
2.Row/Column partition
3.Fox algorithm

Multiple core parallel technologies:
1.PLINQ,
2.TPL,
3.Thread Pool



Pseudo Code of Fox algorithm:

Partitioned matrix A, B to blocks

For each iteration i:

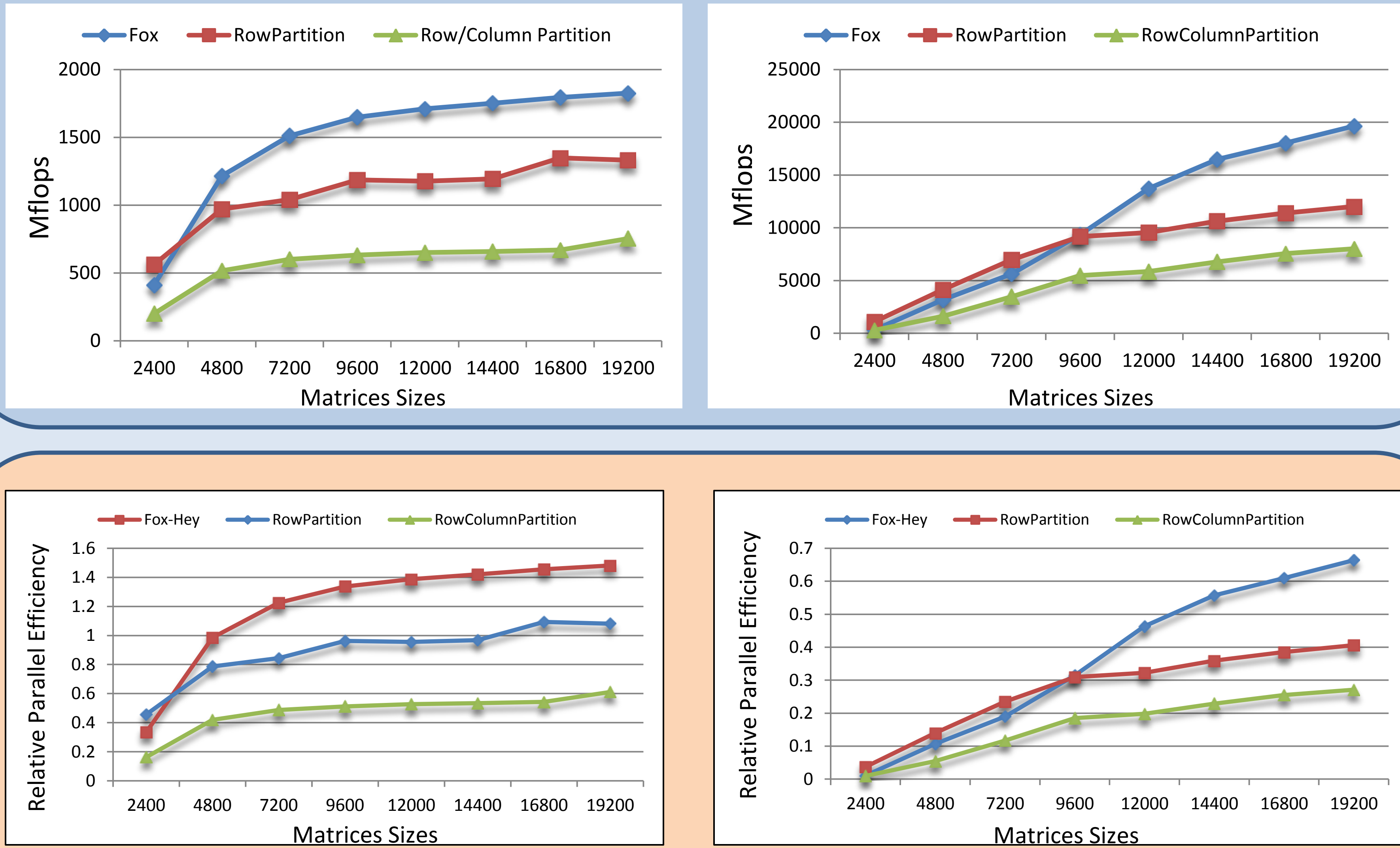
1) broadcast matrix A block (k,j) to row k

2) compute matrix C blocks, and add the partial results to the previous result of matrix C block

3) roll-up matrix B block by column

INDIANA UNIVERSITY

Matrix multiplication performance results
1core on 16 nodes V.S 24 cores on 16 nodes

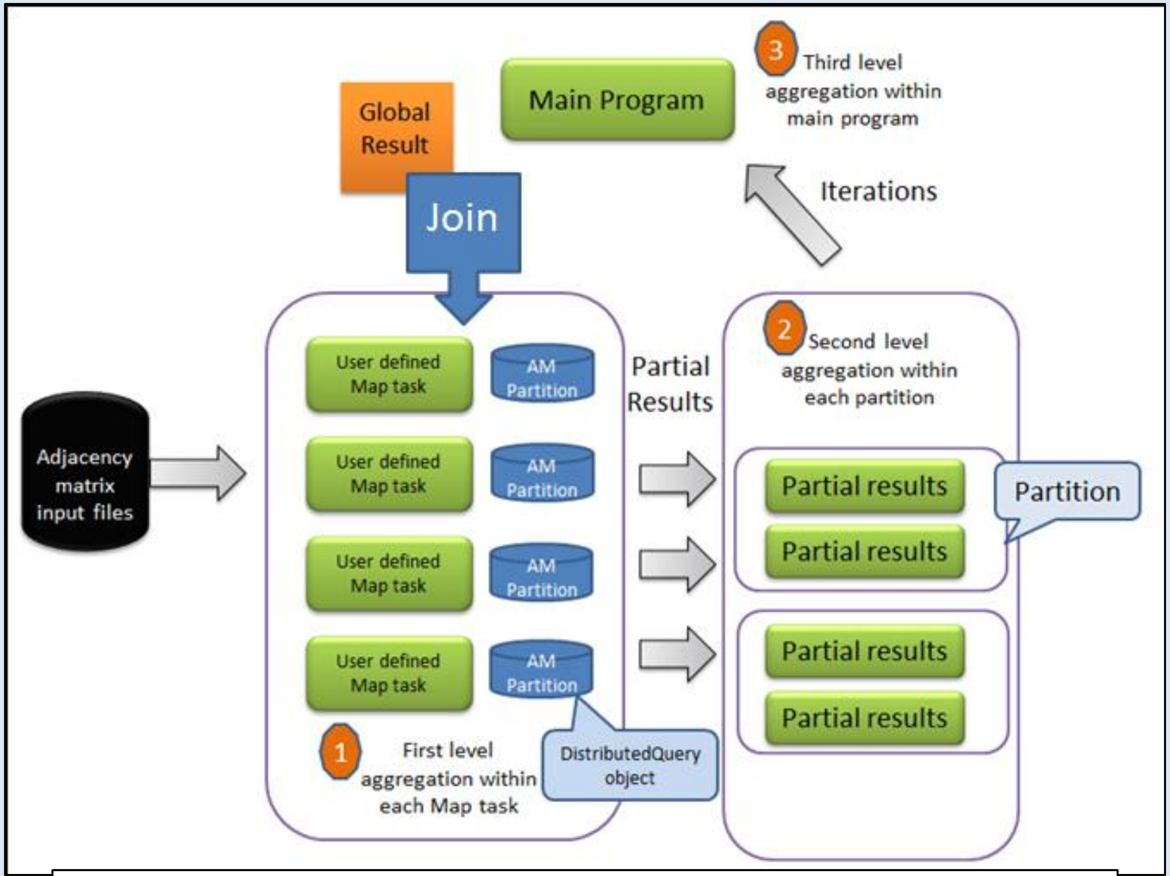


PageRank

Three distributed grouped aggregation approaches

1. Naïve aggregation
2. Hierarchical aggregation
3. Aggregation Tree

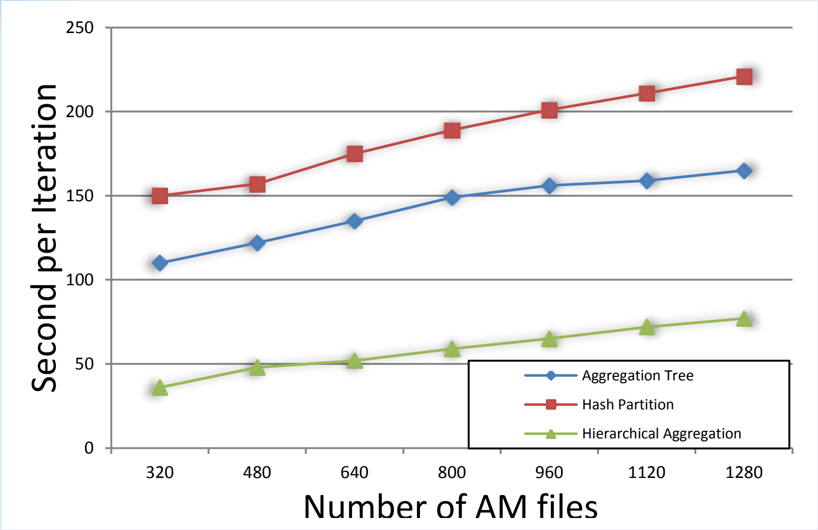
Foreach iteration
{
1. join **edges** with **ranks**
2. distribute **ranks** on **edges**
3. groupBy edge destination
4. aggregate into **ranks**
}

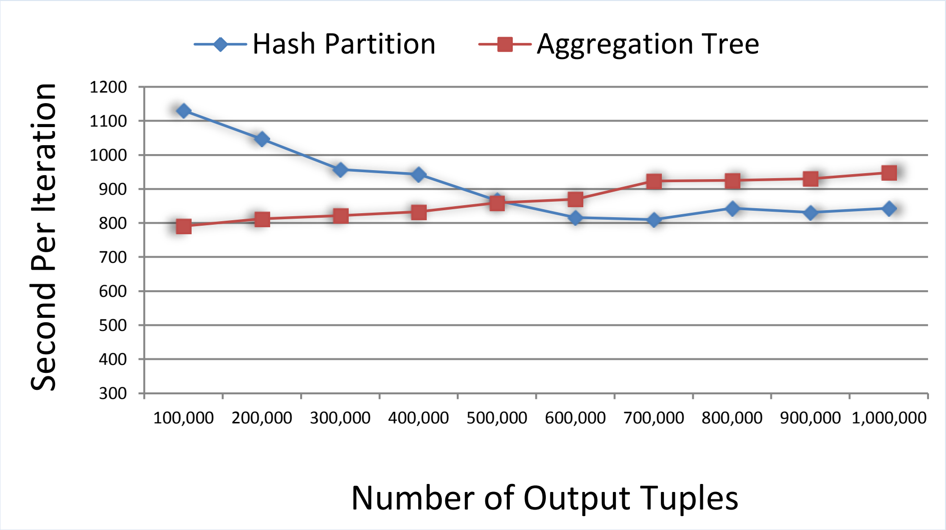


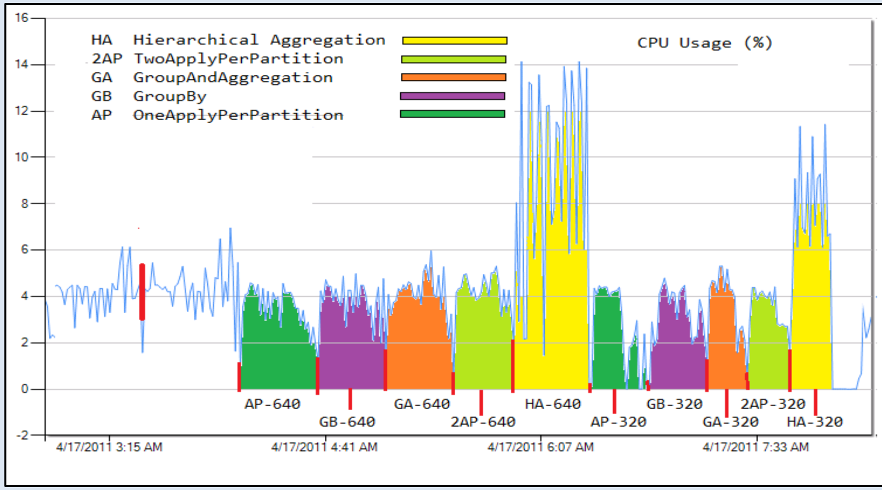
PageRank with hierarchical aggregation approach

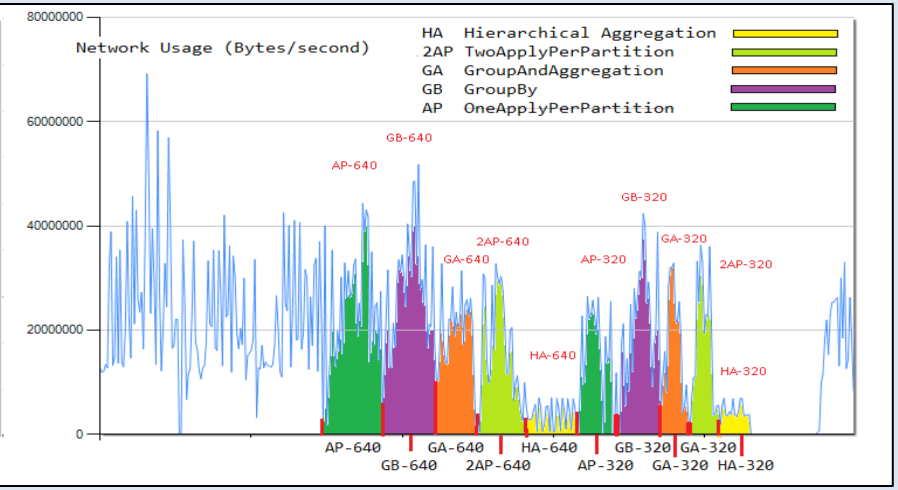
[1] Pagerank Algorithm, http://en.wikipedia.org/wiki/PageRank
[2] ClueWeb09 Data Set, http://boston.lti.cs.cmu.edu/Data/clueweb09/

PageRank performance results
Naïve aggregation v.s Aggregation Tree









Conclusion

- We investigated the usability and performance of using DryadLINQ to develop three applications: SW-G, Matrix Multiply, PageRank. And we abstracted three design patterns: pleasingly parallel programming pattern, hybrid parallel programming pattern, distributed aggregation pattern
- Usability
  - It is easy to tune task granularity with DryadLINQ data model and interfaces
  - The unified data model and interface enable developers to easily utilize the parallelism of single-core, multi-core and multi-node environments.
  - DryadLINQ provide optimized distributed aggregation approaches, and the choice of approaches have materialized effect on the performance
- Performance
  - The parallel MM algorithm scale up for large matrices sizes.
  - Distributed aggregation optimization can speed up the program by 30% than naïve approach

Question?

- Thank you!

Outline

- Introduction
  - Big Data Challenge
  - MapReduce
  - DryadLINQ CTP
- Programming Model Using DryadLINQ
  - Pleasingly
  - Hybrid
  - Distributed Grouped Aggregation
- Implementation
- Discussion and Conclusion

Workflow of Dryad job