A Software Stack for Distributed Data-Parallel Computing

Yuan Yu
Microsoft Research Silicon Valley

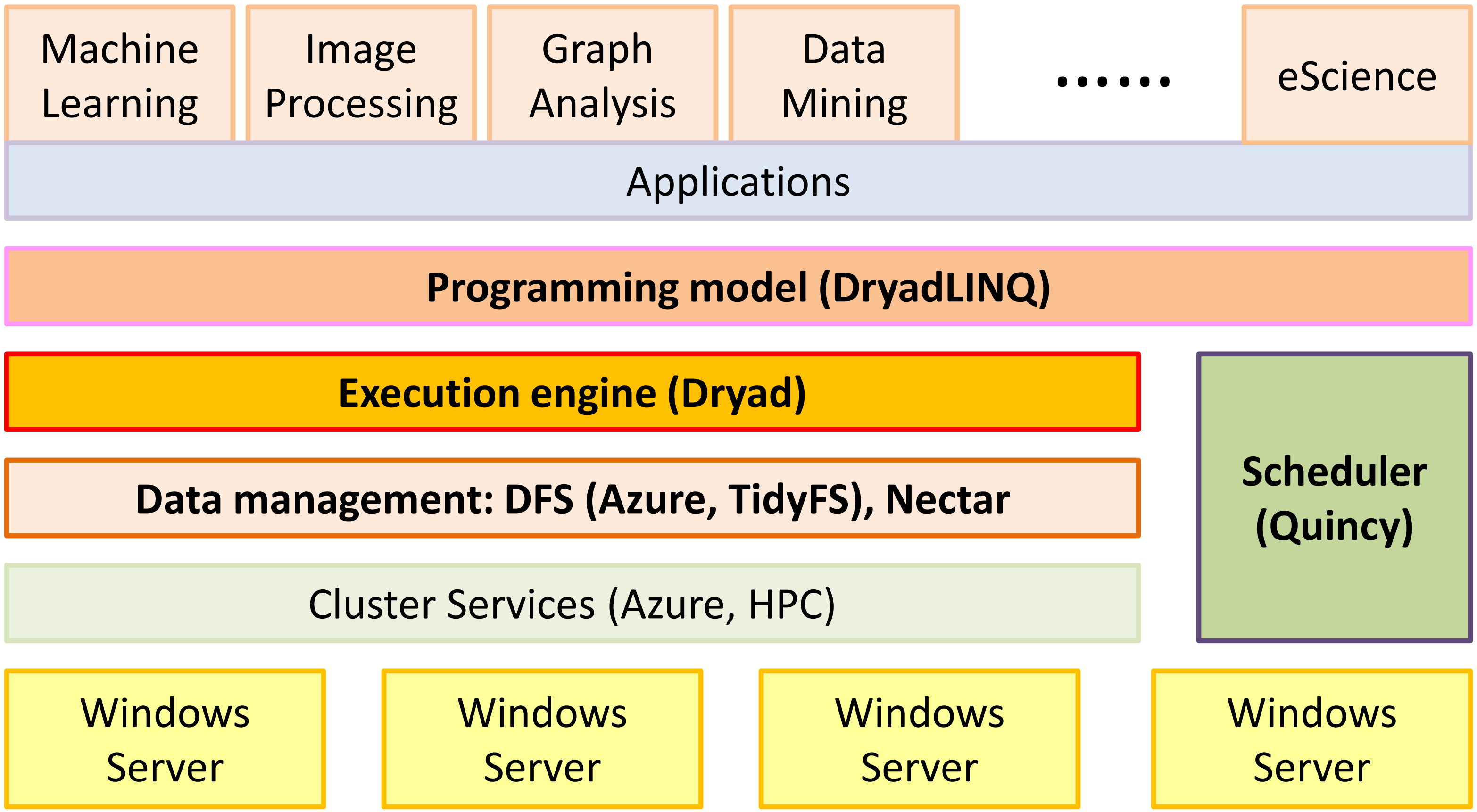
The Programming Model

- Provide a sequential, single machine programming abstraction
  - Same program runs on single-core, multi-core, or cluster
- Preserve the existing programming environments
  - Modern languages (C# and Java) are very good
    - Expressive language and data model
    - Strong static typing, GC, generics, …
  - Modern IDEs (Visual Studio and Eclipse) are very good
    - Great debugging and library support
  - Existing code can be easily reused

LINQ

- Microsoft's Language INtegrated Query
  - Available in .NET3.5 and Visual Studio 2008
- A set of operators to manipulate datasets in .NET
  - Support traditional relational operators
    - Select, Join, GroupBy, Aggregate, etc.
  - Integrated into .NET programming languages
    - Programs can invoke operators
    - Operators can invoke arbitrary .NET functions
- Data model
  - Data elements are strongly typed .NET objects
  - Much more expressive than relational tables
    - For example, nested data structures

Software Stack

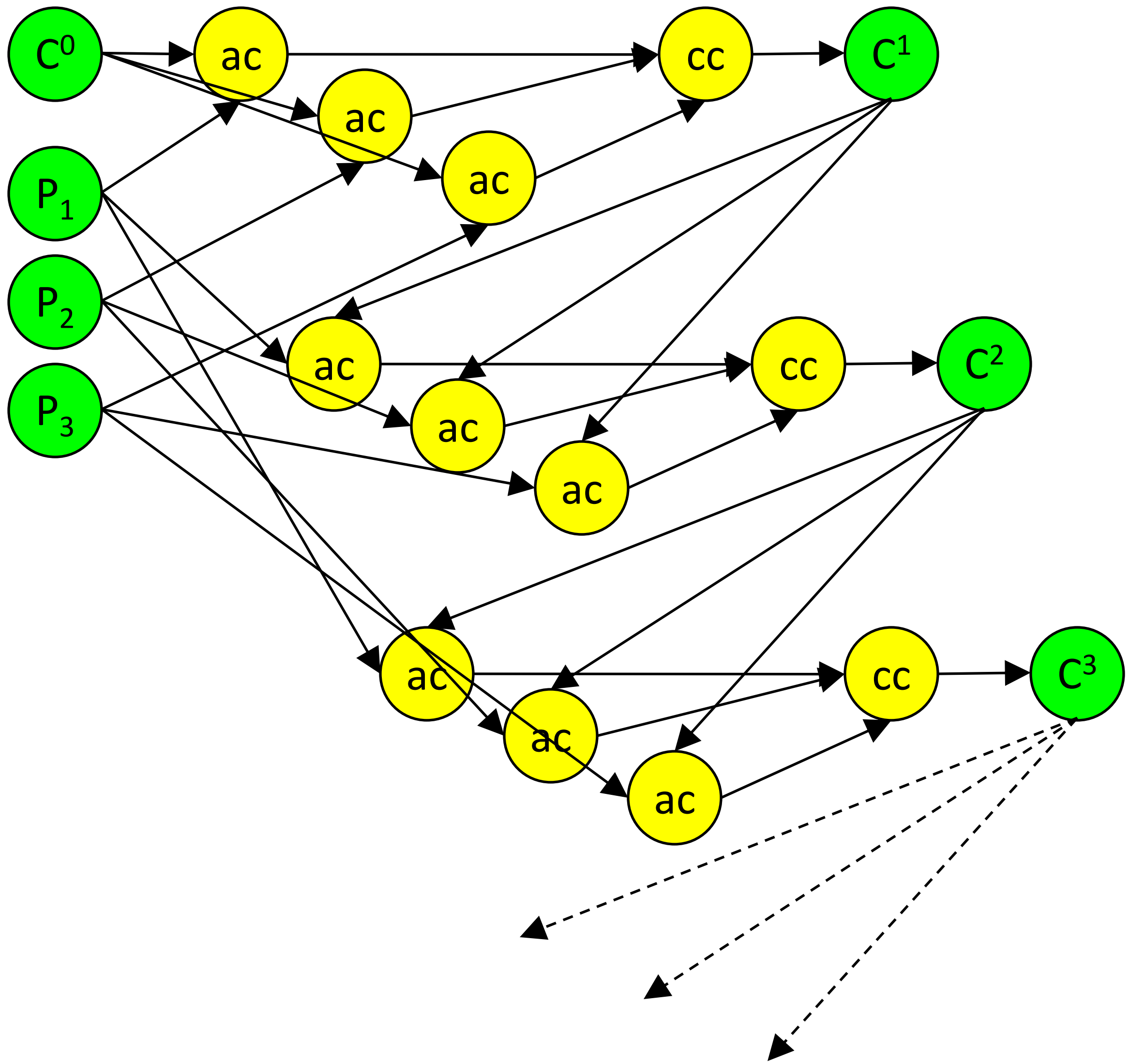


K-means in DryadLINQ

```
public class Vector {
   public double[] elems;

   [Associative]
   public static Vector operator +(Vector v1, Vector v2) {  … }

   public static Vector operator -(Vector v1, Vector v2) { … }

   public double Norm2() { …}
}
```

```
public static Vector NearestCenter(Vector v, IEnumerable<Vector> centers) {
    return centers.Aggregate((r, c) => (r - v).Norm2() < (c - v).Norm2() ? r : c);
}

public static IQueryable<Vector> Step(IQueryable<Vector> vectors, IQueryable<Vector> centers) {
    return vectors.GroupBy(v => NearestCenter(v, centers))
                        .Select(group => group.Aggregate((x,y) => x + y) / group.Count());
}

var vectors = PartitionedTable.Get<Vector>("dfs://vectors.pt");
var centers = vectors.Take(100);
for (int i = 0; i < 10; i++) {
    centers = Step(vectors, centers);
}
centers.ToPartitionedTable<Vector>(“dfs://centers.pt”);
```

Dryad Execution Graph



Availability

- Freely available for academic use and commercial evaluation
  - **http://connect.microsoft.com/DryadLINQ**
  - Only a subset of the stack
- Productization of the stack is under way
  - Transferred to our technical computing team
  - CTP by this November
  - RTM in 2011 running on top of HPC

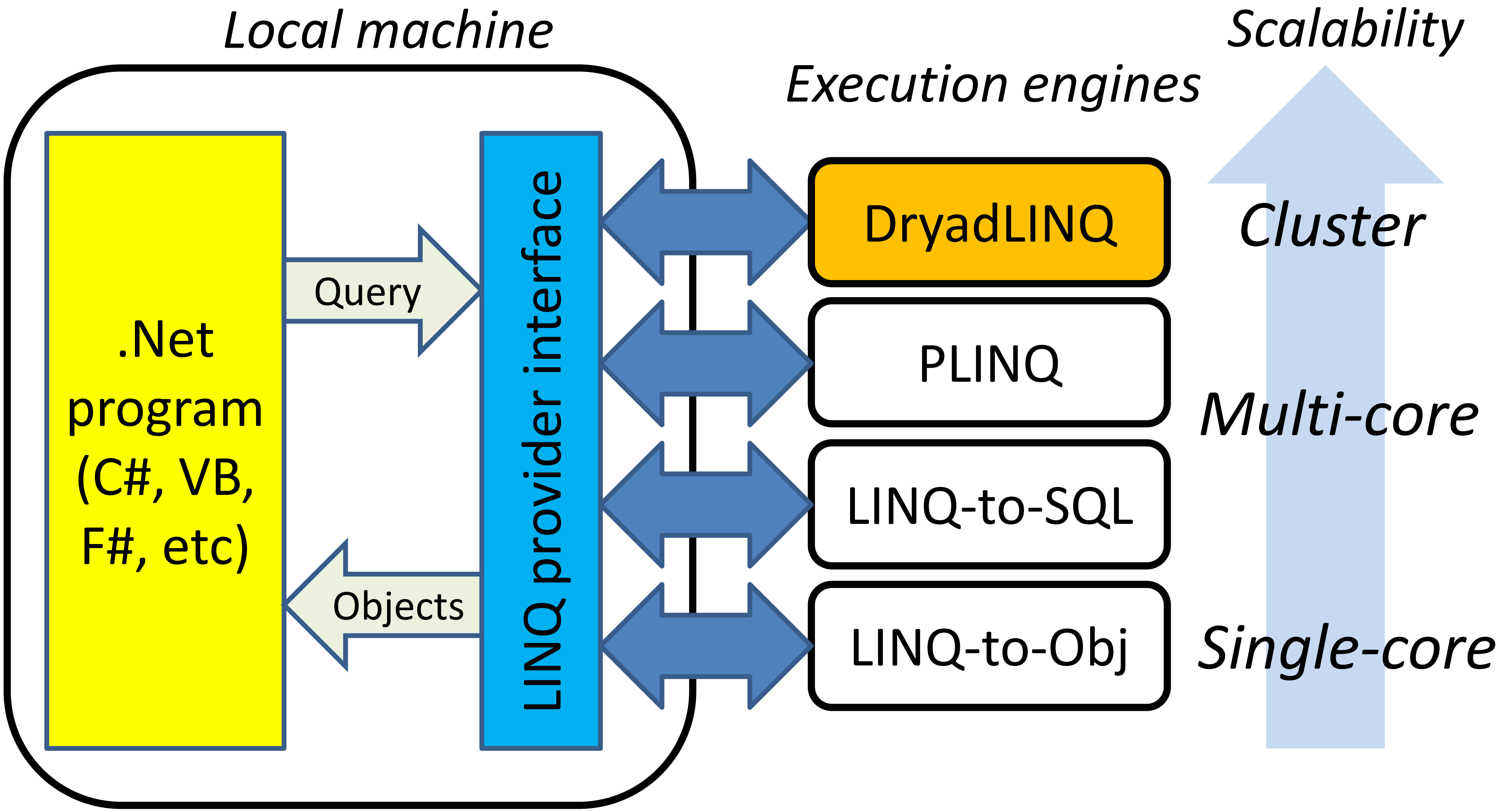
Research Papers

1. Dryad: Distributed Data-Parallel Programs from Sequential Building Blocks (EuroSys'07)
2. DryadLINQ: A System for General-Purpose Distributed Data-Parallel Computing Using a High-Level Language (OSDI'08)
3. Quincy: Fair scheduling for distributed computing clusters (SOSP'09)
4. Distributed Aggregation for Data-Parallel Computing: Interfaces and Implementations (SOSP'09)
5. Nectar: Automatic Management of Data and Computation in Data Centers (OSDI'10)

Language Integration Is Good

- Preserve an existing programming environment
  - Single unified data model and programming language
  - Direct access to IDE and libraries
  - Familiar to the developers
- Simpler than SQL programming
  - As easy for simple queries
  - Easier to use for even moderately complex queries
    - No embedded languages
- FlumeJava (Google) and Spark (Berkeley) followed with the same approach

LINQ Framework Is Good



Decoupling of Dryad and DryadLINQ

- Separation of concerns
  - Dryad layer concerns scheduling and fault-tolerance
  - DryadLINQ layer concerns the programming model and the parallelization of programs
  - Result: efficient and expressive execution engine and programming model
- Different from the MapReduce/Hadoop approach
  - A single abstraction for both programming model and execution engine
  - Result: very simple, but very restricted execution engine and language

Cluster Resources Are Poorly Managed

- A large fraction of computations are redundant
- A lot of datasets are either obsolete or seldom used

Computation

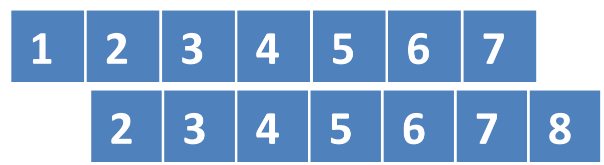
PROBLEM: Redundant Computation

- Programs share sub-computations

  X.Select(F)
  X.Select(F).Where(…)

- Programs share partial input datasets

  X.Select(F)
  (X+X’).Select(F)



SOLUTION: Caching

- Cache the results of *popular* sub-computations
- Rewrite user programs to use cache

Storage

PROBLEM: Unused data occupying space

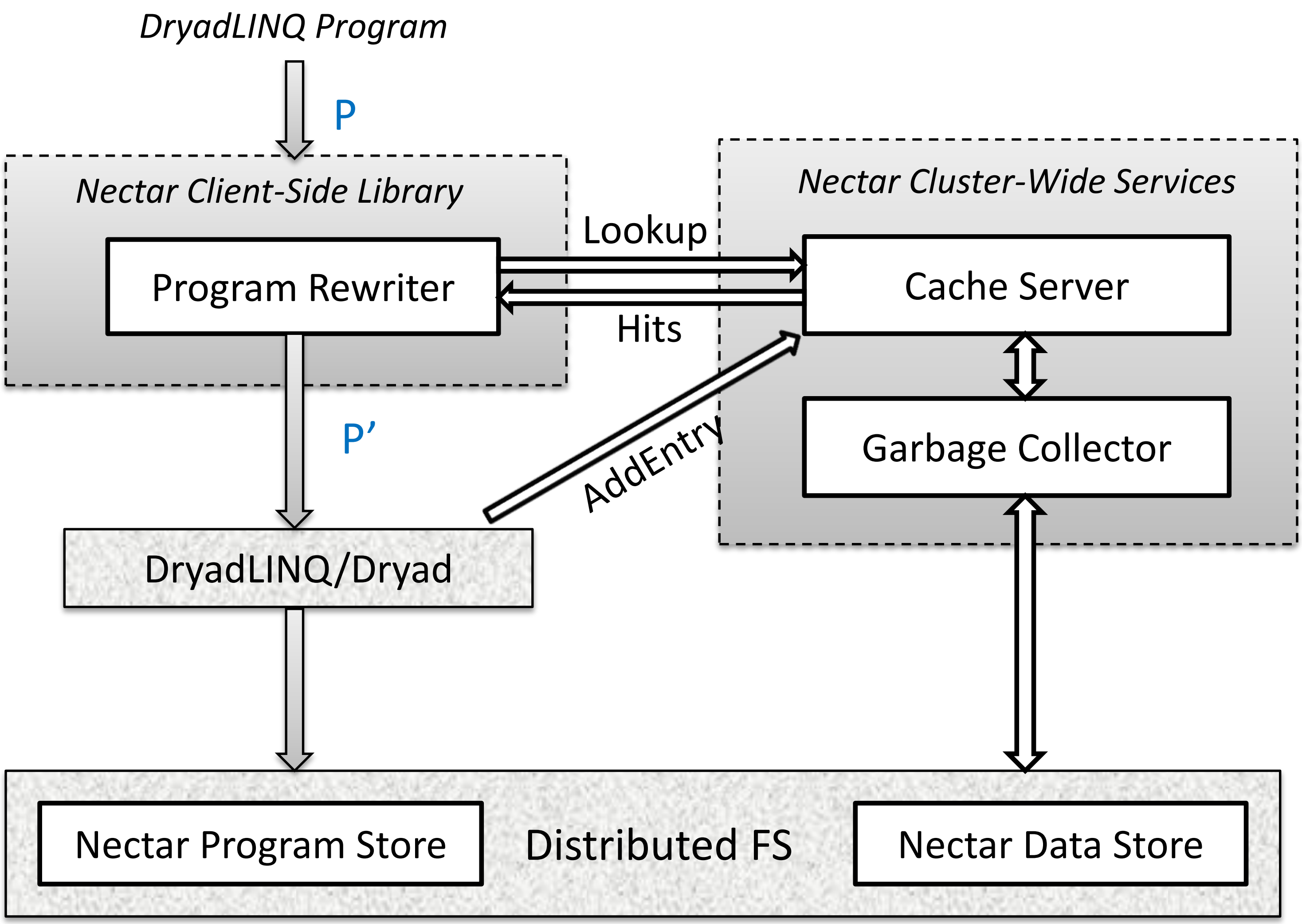
SOLUTION: Automatically manage *derived* datasets

- Divide data into primary and derived
  - Primary: Imported from external sources
  - Derived: Generated by computations
- Delete the derived datasets of the least value
- Recreate a deleted dataset by re-execution
  - Keep the programs of the derived datasets
  - Rerun its program if a dataset is needed after deletion

Program Analysis Is Lacking

- The main sources of difficulty
  - Complicated data model
  - User-defined functions all over the places
- Areas heavily depend on program analysis
  - Many query optimizations
  - Computation caching
  - Purity checking
  - Enforcement of program properties for security and privacy mechanisms
  - Debugging and verification
  - ……

Nectar



DryadLINQ Program
P
Nectar Client-Side Library
Program Rewriter
Lookup
Hits
Nectar Cluster-Wide Services
Cache Server
Garbage Collector
P'
AddEntry
DryadLINQ/Dryad
Nectar Program Store
Distributed FS
Nectar Data Store

System Components

- Program rewriter
  - Rewrite programs to use cache
- Static program dependency analyzer
  - Used to compute a unique fingerprint of a program
- Datacenter-wide cache server
  - Cache popular computations
  - Track usage/cost of cache entries (and hence deriveds)
- Datacenter-wide garbage collector
  - Garbage collect deriveds based on usage/cost
- Program store
  - Store programs so that deriveds can be reproduced