DryadLINQ

A System for General-Purpose Distributed Data-Parallel Computing Using a High-Level Language

Mikołaj Murasik

Faculty of Mathematics, Informatics and Mechanics
Warsaw Univeristy

24-11-2010

Based on the article by Y. Yu, M. Isard, D. Fetterly, M. Budiu, U. Erlingsson, P. K. Gunda and J. Currey

research.microsoft.com/pubs/70861/DryadLINQ-osdi.pdf

What is the current situation?

- emergence of large-scale internet services depending on clusters

What is the current situation?

- emergence of large-scale internet services depending on clusters
- future advances in local computing power will come from increasing the number of cores on a chip rather than improving the speed

What is the current situation?

- emergence of large-scale internet services depending on clusters
- future advances in local computing power will come from increasing the number of cores on a chip rather than improving the speed
- problems concerning distributed systems are complex and difficult to solve

What is the current situation?

- emergence of large-scale internet services depending on clusters
- future advances in local computing power will come from increasing the number of cores on a chip rather than improving the speed
- problems concerning distributed systems are complex and difficult to solve
- simplest way to achieve scalable performance is to exploit data parallelism

What do we need?

- simplicity of the programming model

What do we need?

- simplicity of the programming model
- reliability, efficiency and scalability of the applications

What do we need?

- simplicity of the programming model
- reliability, efficiency and scalability of the applications
- resources being in a single administrative domain, connected using a known, high-performance communication topology, under centralized management and control

What do we need?

- simplicity of the programming model
- reliability, efficiency and scalability of the applications
- resources being in a single administrative domain, connected using a known, high-performance communication topology, under centralized management and control
- simple sidestepping of typical problems: latency, unreliable networks, access control, control of resources, ...

How do we do it?

- transparently compile imperative programs in general-purpose language into distributed computations

How do we do it?

- transparently compile imperative programs in general-purpose language into distributed computations
- give the illusion of writing for a single machine

How do we do it?

- transparently compile imperative programs in general-purpose language into distributed computations
- give the illusion of writing for a single machine

Solution

Use a high-level language that does (nearly) everything for you

What is LINQ?

- Language INtegrated Query

What is LINQ?

- Language INtegrated Query
- Microsoft .NET Framework component that adds native data querying capabilities to .NET languages

What does it do?

- a set of .NET constructs (query operators) for manipulating sets and sequences of data items:

What does it do?

- a set of .NET constructs (query operators) for manipulating sets and sequences of data items:
  - Select

What does it do?

- a set of .NET constructs (query operators) for manipulating sets and sequences of data items:
  - Select
  - SelectMany

What does it do?

- a set of .NET constructs (query operators) for manipulating sets and sequences of data items:
  - Select
  - SelectMany
  - Where

What does it do?

- a set of .NET constructs (query operators) for manipulating sets and sequences of data items:
  - Select
  - SelectMany
  - Where
  - Aggregate

What does it do?

- a set of .NET constructs (query operators) for manipulating sets and sequences of data items:
  - Select
  - SelectMany
  - Where
  - Aggregate
  - Join

What does it do?

- a set of .NET constructs (query operators) for manipulating sets and sequences of data items:
  - Select
  - SelectMany
  - Where
  - Aggregate
  - Join
- filters data in arrays, enumerable classes, XML, relational database and third party data sources

What does it do?

- a set of .NET constructs (query operators) for manipulating sets and sequences of data items:
  - Select
  - SelectMany
  - Where
  - Aggregate
  - Join
- filters data in arrays, enumerable classes, XML, relational database and third party data sources
- with LINQ the programmer may express complex computations over datasets while giving the runtime great leeway to decide how these computations should be implemented

What is Dryad?

- ongoing research project at Microsoft Research

What is Dryad?

- ongoing research project at Microsoft Research
- has been in continuous operation for several years

What is Dryad?

- ongoing research project at Microsoft Research
- has been in continuous operation for several years
- general purpose runtime for execution of data parallel applications

What is Dryad?

- ongoing research project at Microsoft Research
- has been in continuous operation for several years
- general purpose runtime for execution of data parallel applications
- takes care of typical DS problems

Dryad's basic characteristics

- no need to take care of fine-grained concurrency control

Dryad's basic characteristics

- no need to take care of fine-grained concurrency control
  - mutexes

Dryad's basic characteristics

- no need to take care of fine-grained concurrency control
  - mutexes
  - critical sections

Dryad's basic characteristics

- no need to take care of fine-grained concurrency control
  - mutexes
  - critical sections
  - ...

Dryad's basic characteristics

- no need to take care of fine-grained concurrency control
  - mutexes
  - critical sections
  - ...
- working on high level of abstraction

Dryad's basic characteristics

- no need to take care of fine-grained concurrency control
  - mutexes
  - critical sections
  - ...
- working on high level of abstraction
- coarse-grained concurrency

Dryad's basic characteristics

- no need to take care of fine-grained concurrency control
  - mutexes
  - critical sections
  - ...
- working on high level of abstraction
- coarse-grained concurrency
- spawning subroutines across many machines

How does Dryad work?

- application written for Dryad is modeled as a DAG

How does Dryad work?

- application written for Dryad is modeled as a DAG
- the DAG defines the dataflow of the application

How does Dryad work?

- application written for Dryad is modeled as a DAG
- the DAG defines the dataflow of the application
- vertices defines the operations that are to be performed on the data

How does Dryad work?

- application written for Dryad is modeled as a DAG
- the DAG defines the dataflow of the application
- vertices defines the operations that are to be performed on the data
- *computational vertices* are written using sequential constructs, devoid any concurrency or mutual exclusion semantics

How does Dryad work?

But:

- a developer must instruct Dryad how to spawn the work

How does Dryad work?

But:

- a developer must instruct Dryad how to spawn the work
- programming on a higher level of abstraction would be welcome

How does Dryad work?

But:

- a developer must instruct Dryad how to spawn the work
- programming on a higher level of abstraction would be welcome

So...

...Dryad is used as an underlaying layer for higher-level systems

What can a developer forget about?

- everything is planned by the *Job Manager*

What can a developer forget about?

- everything is planned by the *Job Manager*
- Dryad runtime parallelizes the dataflow graph by distributing the computational vertices across various execution engines

What can a developer forget about?

- everything is planned by the *Job Manager*
- Dryad runtime parallelizes the dataflow graph by distributing the computational vertices across various execution engines
- Dryad runtime schedules the computational vertices on the available hardware, without any explicit intervention by the developer

What can a developer forget about?

- everything is planned by the *Job Manager*
- Dryad runtime parallelizes the dataflow graph by distributing the computational vertices across various execution engines
- Dryad runtime schedules the computational vertices on the available hardware, without any explicit intervention by the developer
- the flow of data between one computational vertex to another is implemented by using communication "channels" between the vertices

What can a developer forget about?

- everything is planned by the *Job Manager*
- Dryad runtime parallelizes the dataflow graph by distributing the computational vertices across various execution engines
- Dryad runtime schedules the computational vertices on the available hardware, without any explicit intervention by the developer
- the flow of data between one computational vertex to another is implemented by using communication "channels" between the vertices
  - TCP/IP streams

What can a developer forget about?

- everything is planned by the *Job Manager*
- Dryad runtime parallelizes the dataflow graph by distributing the computational vertices across various execution engines
- Dryad runtime schedules the computational vertices on the available hardware, without any explicit intervention by the developer
- the flow of data between one computational vertex to another is implemented by using communication "channels" between the vertices
  - TCP/IP streams
  - shared memory

What can a developer forget about?

- everything is planned by the *Job Manager*
- Dryad runtime parallelizes the dataflow graph by distributing the computational vertices across various execution engines
- Dryad runtime schedules the computational vertices on the available hardware, without any explicit intervention by the developer
- the flow of data between one computational vertex to another is implemented by using communication "channels" between the vertices
  - TCP/IP streams
  - shared memory
  - temporary files

Dryad's workflow

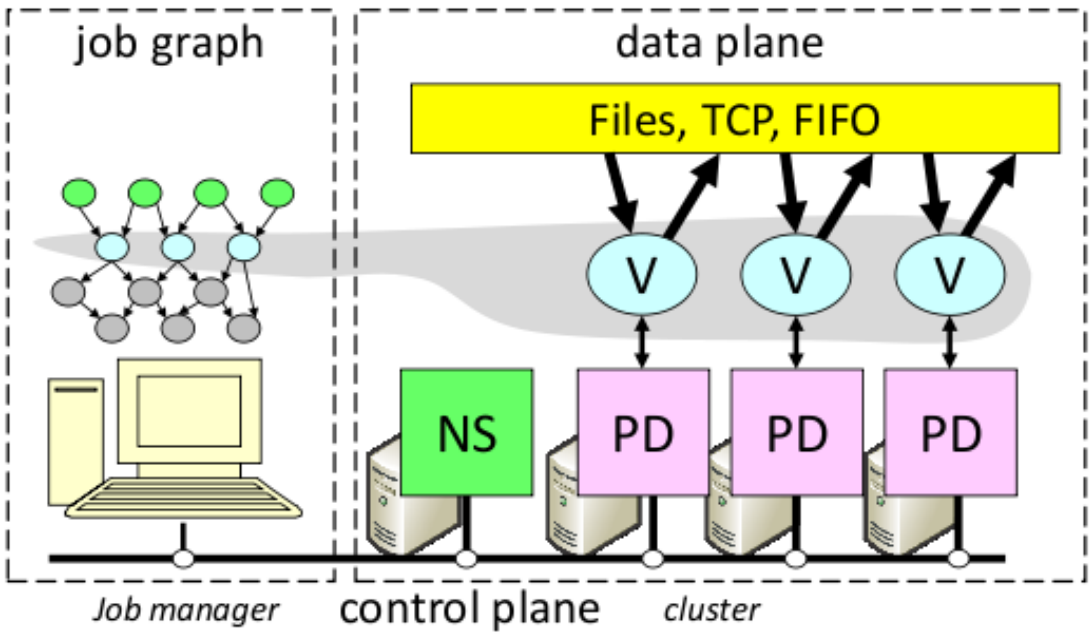

Figure 1: *Dryad system architecture. NS is the name server which maintains the cluster membership. The job manager is responsible for spawning vertices (V) on available computers with the help of a remote-execution and monitoring daemon (PD). Vertices exchange data through files, TCP pipes, or shared-memory channels. The grey shape indicates the vertices in the job that are currently running and the correspondence with the job execution graph.*

Dryad's Job Manager

Job Manager is responsible for:

- instantiating a job's dataflow graph

Dryad's Job Manager

Job Manager is responsible for:

- instantiating a job's dataflow graph
- scheduling processes on cluster computers

Dryad's Job Manager

Job Manager is responsible for:

- instantiating a job's dataflow graph
- scheduling processes on cluster computers
- providing fault-tolerance by re-executing failed or slow processes

Dryad's Job Manager

Job Manager is responsible for:

- instantiating a job's dataflow graph
- scheduling processes on cluster computers
- providing fault-tolerance by re-executing failed or slow processes
- monitoring the job and collecting statistics

Dryad's Job Manager

Job Manager is responsible for:

- instantiating a job's dataflow graph
- scheduling processes on cluster computers
- providing fault-tolerance by re-executing failed or slow processes
- monitoring the job and collecting statistics
- transforming the job graph dynamically according to user-supplied policies

Dryad's Job Manager

Job Manager is responsible for:

- instantiating a job's dataflow graph
- scheduling processes on cluster computers
- providing fault-tolerance by re-executing failed or slow processes
- monitoring the job and collecting statistics
- transforming the job graph dynamically according to user-supplied policies

But:

A cluster is typically controlled by a task scheduler, separate from Dryad, which manages a batch queue of jobs and executes a few at a time subject to cluster policy

What is DryadLINQ?

DryadLINQ...

...is a system and a set of language extensions that enable a new programming model for large scale distributed computing

What is DryadLINQ?

- designed to make it easy for a variety of developers to compute effectively on large amounts of data

What is DryadLINQ?

- designed to make it easy for a variety of developers to compute effectively on large amounts of data
- programs written as imperative or declarative operations on datasets within a traditional high-level language

What is DryadLINQ?

- designed to make it easy for a variety of developers to compute effectively on large amounts of data
- programs written as imperative or declarative operations on datasets within a traditional high-level language
- uses expressive data model of strongly typed .NET objects

What is DryadLINQ?

- designed to make it easy for a variety of developers to compute effectively on large amounts of data
- programs written as imperative or declarative operations on datasets within a traditional high-level language
- uses expressive data model of strongly typed .NET objects
- objects in DryadLINQ datasets can be of any .NET type, making it easy to compute with data such as image patches, vectors, and matrices

Why not SQL?

SQL adopts a very restrictive type system. In addition, the declarative "query-oriented" nature of SQL makes it difficult to express common programming patterns such as iteration.

Why not SQL?

SQL adopts a very restrictive type system. In addition, the declarative "query-oriented" nature of SQL makes it difficult to express common programming patterns such as iteration.

- these make SQL unsuitable for tasks such as machine learning, content parsing, and web-graph analysis that increasingly must be run on very large datasets

Why not MapReduce?

- MapReduce system adopted a radically simplified programming abstraction

Why not MapReduce?

- MapReduce system adopted a radically simplified programming abstraction
- lack of any type-system support or integration between the MapReduce stages requires programmers to explicitly keep track of objects passed between these stages

Why not MapReduce?

- MapReduce system adopted a radically simplified programming abstraction
- lack of any type-system support or integration between the MapReduce stages requires programmers to explicitly keep track of objects passed between these stages
- complications with long-term maintenance and re-use of software components

Why not others?

- similar problems to aforementioned

Why not others?

- similar problems to aforementioned
- DryadLINQ is the most flexible

Virtualized expression plans

- unlike those in databases

Virtualized expression plans

- unlike those in databases
- planner allocates resources independent of the actual cluster used for execution

Virtualized expression plans

- unlike those in databases
- planner allocates resources independent of the actual cluster used for execution
- DryadLINQ can run plans requiring many more steps than the instantaneously available computation resources would permit

Virtualized expression plans

- unlike those in databases
- planner allocates resources independent of the actual cluster used for execution
- DryadLINQ can run plans requiring many more steps than the instantaneously available computation resources would permit
- computational resources can change dynamically, e.g. due to faults

Virtualized expression plans

- unlike those in databases
- planner allocates resources independent of the actual cluster used for execution
- DryadLINQ can run plans requiring many more steps than the instantaneously available computation resources would permit
- computational resources can change dynamically, e.g. due to faults
- we have an extra degree of freedom in buffer management

Virtualized expression plans

- unlike those in databases
- planner allocates resources independent of the actual cluster used for execution
- DryadLINQ can run plans requiring many more steps than the instantaneously available computation resources would permit
- computational resources can change dynamically, e.g. due to faults
- we have an extra degree of freedom in buffer management

But:

virtualization requires intermediate results to be stored to persistent media, potentially increasing computation latency

Where can we use DryadLINQ?

- DryadLINQ compiles LINQ programs into distributed computations running on the Dryad cluster-computing infrastructure

Where can we use DryadLINQ?

- DryadLINQ compiles LINQ programs into distributed computations running on the Dryad cluster-computing infrastructure
- data model and serialization for Dryad are provided by higher-level layer: DryadLINQ

How does it work?

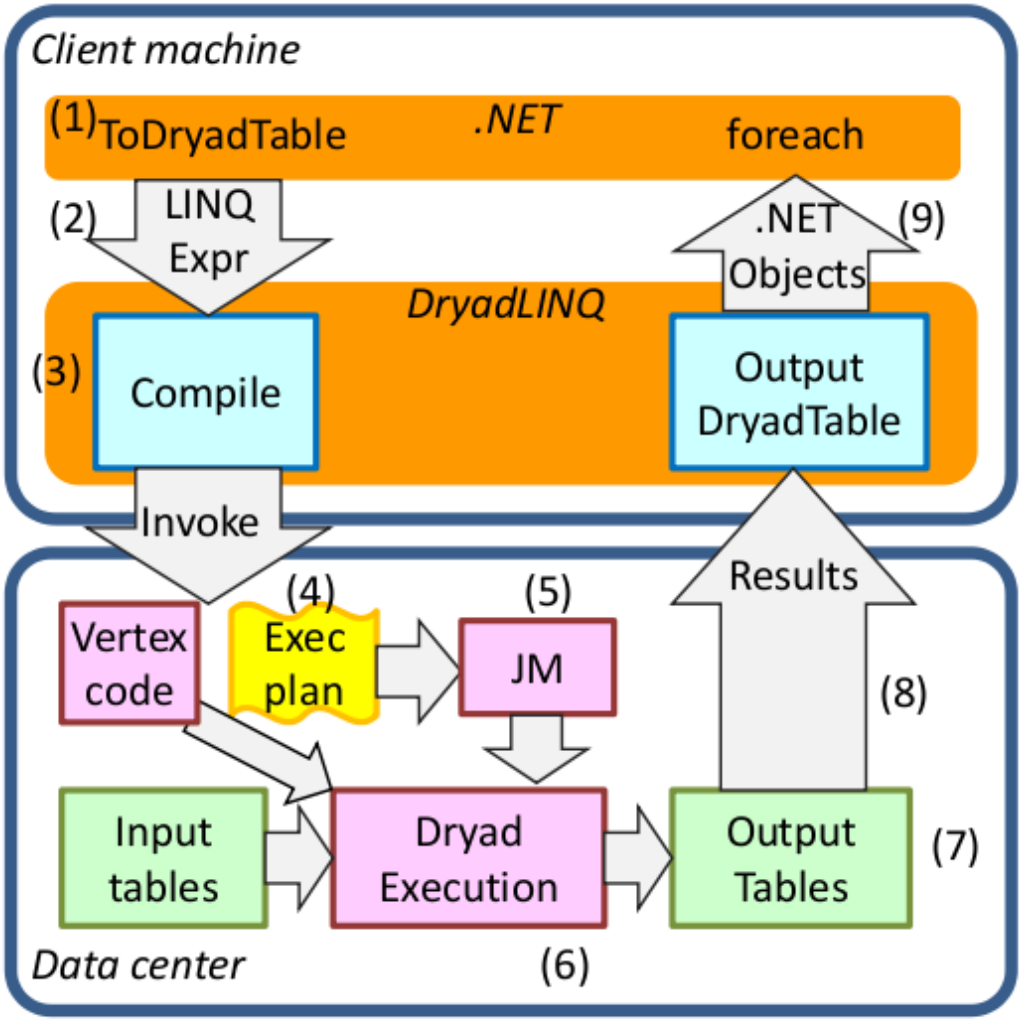

Figure 2: *LINQ-expression execution in DryadLINQ.*

How does it work?

1. a .NET user application runs. It creates a DryadLINQ expression object. Because of LINQ's deferred evaluation, the actual execution of the expression has not occurred

How does it work?

1. a .NET user application runs. It creates a DryadLINQ expression object. Because of LINQ's deferred evaluation, the actual execution of the expression has not occurred
2. the application calls ToDryadTable triggering a data-parallel execution; the expression object is handed to DryadLINQ

How does it work?

1. a .NET user application runs. It creates a DryadLINQ expression object. Because of LINQ's deferred evaluation, the actual execution of the expression has not occurred
2. the application calls ToDryadTable triggering a data-parallel execution; the expression object is handed to DryadLINQ
3. DryadLINQ compiles the LINQ expression into a distributed Dryad execution plan. It performs: (a) the decomposition of the expression into subexpressions, each to be run in a separate Dryad vertex; (b) the generation of code and static data for the remote Dryad vertices; and (c) the generation of serialization code for the required data types

How does it work?

4. DryadLINQ invokes a custom, DryadLINQ specific, Dryad job manager. The job manager may be executed behind a cluster firewall

How does it work?

4. DryadLINQ invokes a custom, DryadLINQ specific, Dryad job manager. The job manager may be executed behind a cluster firewall
5. the job manager creates the job graph using the plan created earlier. It schedules and spawns the vertices as resources become available

How does it work?

4. DryadLINQ invokes a custom, DryadLINQ specific, Dryad job manager. The job manager may be executed behind a cluster firewall
5. the job manager creates the job graph using the plan created earlier. It schedules and spawns the vertices as resources become available
6. each Dryad vertex executes a vertex-specific program

How does it work?

4. DryadLINQ invokes a custom, DryadLINQ specific, Dryad job manager. The job manager may be executed behind a cluster firewall
5. the job manager creates the job graph using the plan created earlier. It schedules and spawns the vertices as resources become available
6. each Dryad vertex executes a vertex-specific program
7. when the Dryad job completes successfully it writes the data to the output table(s)

How does it work?

8. The job manager process terminates, and it returns control back to DryadLINQ. DryadLINQ creates the local DryadTable objects encapsulating the outputs of the execution. These objects may be used as inputs to subsequent expressions in the user program. Data objects within a DryadTable output are fetched to the local context only if explicitly dereferenced

How does it work?

8. The job manager process terminates, and it returns control back to DryadLINQ. DryadLINQ creates the local DryadTable objects encapsulating the outputs of the execution. These objects may be used as inputs to subsequent expressions in the user program. Data objects within a DryadTable output are fetched to the local context only if explicitly dereferenced
9. Control returns to the user application. The iterator interface over a DryadTable allows the user to read its contents as .NET objects

How does it work?

8. The job manager process terminates, and it returns control back to DryadLINQ. DryadLINQ creates the local DryadTable objects encapsulating the outputs of the execution. These objects may be used as inputs to subsequent expressions in the user program. Data objects within a DryadTable output are fetched to the local context only if explicitly dereferenced
9. Control returns to the user application. The iterator interface over a DryadTable allows the user to read its contents as .NET objects
10. The application may generate subsequent DryadLINQ expressions, to be executed by a repetition of previous steps

General observation

DryadLINQ...

...programs may be written in any .NET language: C#, F#, ...

LINQ from a developer's perspective

- interfaces: `IEnumerable<T>`, `IQueryable<T>`

LINQ from a developer's perspective

- interfaces: `IEnumerable<T>`, `IQueryable<T>`
- we do not care how they are implemented

LINQ from a developer's perspective

- interfaces: `IEnumerable<T>`, `IQueryable<T>`
- we do not care how they are implemented
- DryadLINQ composes all LINQ expressions into `IQueryable` objects and defers evaluation until the result is needed, at which point the expression graph within the `IQueryable` is optimized and executed in its entirety on the cluster

DryadLINQ and LINQ

- DryadLINQ preserves the LINQ programming model and extends it to data-parallel programming by defining a small set of new operators and datatypes

DryadLINQ and LINQ

- DryadLINQ preserves the LINQ programming model and extends it to data-parallel programming by defining a small set of new operators and datatypes
- DryadLINQ data model is a distributed implementation of LINQ collections

DryadLINQ and LINQ

- DryadLINQ preserves the LINQ programming model and extends it to data-parallel programming by defining a small set of new operators and datatypes
- DryadLINQ data model is a distributed implementation of LINQ collections
- dataset may still contain arbitrary .NET types, but each DryadLINQ dataset is in general distributed across the computers of a cluster, partitioned into disjoint pieces

DryadLINQ and LINQ

- DryadLINQ preserves the LINQ programming model and extends it to data-parallel programming by defining a small set of new operators and datatypes
- DryadLINQ data model is a distributed implementation of LINQ collections
- dataset may still contain arbitrary .NET types, but each DryadLINQ dataset is in general distributed across the computers of a cluster, partitioned into disjoint pieces

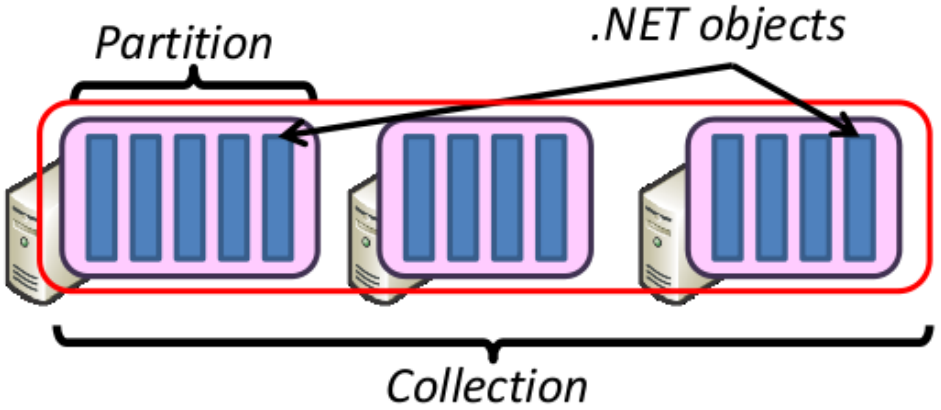

Figure 4: *The DryadLINQ data model: strongly-typed collections of .NET objects partitioned on a set of computers.*

Partitioning strategies

- hash-partitioning

Partitioning strategies

- hash-partitioning
- range-partitioning

Partitioning strategies

- hash-partitioning
- range-partitioning
- round-robin

Partitioning strategies

- hash-partitioning
- range-partitioning
- round-robin

Dataset partitioning...

...is managed transparently by the system unless the programmer explicitly overrides the optimizer's choices

DryadLINQ objects

- the inputs and outputs of a DryadLINQ computation are represented by objects of type `DryadTable<T>`, which is a subtype of `IQueryable<T>`

DryadLINQ objects

- the inputs and outputs of a DryadLINQ computation are represented by objects of type `DryadTable<T>`, which is a subtype of `IQueryable<T>`
- subtypes of `DryadTable<T>` support underlying storage providers that include distributed filesystems, collections of NTFS files, and sets of SQL tables

DryadLINQ objects

- the inputs and outputs of a DryadLINQ computation are represented by objects of type `DryadTable<T>`, which is a subtype of `IQueryable<T>`
- subtypes of `DryadTable<T>` support underlying storage providers that include distributed filesystems, collections of NTFS files, and sets of SQL tables
- DryadTable objects may include metadata read from the file system describing table properties such as schemas for the data items contained in the table, and partitioning schemes

How can this work?

Beware!

All the functions called in DryadLINQ expressions must be side-effect free!

How can this work?

Beware!

All the functions called in DryadLINQ expressions must be side-effect free!

- shared objects can be referenced and read freely and will be automatically serialized and distributed where necessary

How can this work?

Beware!

All the functions called in DryadLINQ expressions must be side-effect free!

- shared objects can be referenced and read freely and will be automatically serialized and distributed where necessary
- if any shared object is modified, the result of the computation is undefined

How can this work?

Beware!

All the functions called in DryadLINQ expressions must be side-effect free!

- shared objects can be referenced and read freely and will be automatically serialized and distributed where necessary
- if any shared object is modified, the result of the computation is undefined

DryadLINQ...

...does *not* currently check or enforce the absence of side-effects

DryadLINQ additional operators

- `HashPartition<T, K>`

DryadLINQ additional operators

- `HashPartition<T, K>`
- `RangePartition<T, K>`

DryadLINQ additional operators

- `HashPartition<T, K>`
- `RangePartition<T, K>`
- `Apply`

DryadLINQ additional operators

- `HashPartition<T, K>`
- `RangePartition<T, K>`
- `Apply`
- `Fork`

DryadLINQ additional operators

- `HashPartition<T, K>`
- `RangePartition<T, K>`
- `Apply`
- `Fork`

These operators...

...may change optimizer's execution plan, which is rarely necessary

Development and system optimizer

- optimizer produces very good automatic plan fairly often

Development and system optimizer

- optimizer produces very good automatic plan fairly often
- developer may add annotations to improve it

Development and system optimizer

- optimizer produces very good automatic plan fairly often
- developer may add annotations to improve it
- it may be particularly necessary with `Apply` and `Fork`

What does the system do?

- DryadLINQ's optimizer is similar do databases' optimizers

What does the system do?

- DryadLINQ's optimizer is similar do databases' optimizers
- static component generates Execution Plan Graph (EPG)

What does the system do?

- DryadLINQ's optimizer is similar do databases' optimizers
- static component generates Execution Plan Graph (EPG)
- dynamic component optimizes the graph at run time

Execution Plan Graph

- a DAG

Execution Plan Graph

- a DAG
- a skeleton of the Dryad data-flow graph to be executed

Execution Plan Graph

- a DAG
- a skeleton of the Dryad data-flow graph to be executed
- optimized before passing to Dryad stage

Execution Plan Graph

- a DAG
- a skeleton of the Dryad data-flow graph to be executed
- optimized before passing to Dryad stage
- annotated with metadata

Execution Plan Graph

- a DAG
- a skeleton of the Dryad data-flow graph to be executed
- optimized before passing to Dryad stage
- annotated with metadata
  - patitioning scheme

Execution Plan Graph

- a DAG
- a skeleton of the Dryad data-flow graph to be executed
- optimized before passing to Dryad stage
- annotated with metadata
  - patitioning scheme
  - ordering information

Execution Plan Graph

- a DAG
- a skeleton of the Dryad data-flow graph to be executed
- optimized before passing to Dryad stage
- annotated with metadata
  - patitioning scheme
  - ordering information
  - compression theme

Execution Plan Graph

- a DAG
- a skeleton of the Dryad data-flow graph to be executed
- optimized before passing to Dryad stage
- annotated with metadata
  - patitioning scheme
  - ordering information
  - compression theme
  - serialization codes

Execution Plan Graph

- a DAG
- a skeleton of the Dryad data-flow graph to be executed
- optimized before passing to Dryad stage
- annotated with metadata
  - patitioning scheme
  - ordering information
  - compression theme
  - serialization codes
  - ...

Metadata

- used in choosing algorithms to execute

Metadata

- used in choosing algorithms to execute
- difficult to propagate

Metadata

- used in choosing algorithms to execute
- difficult to propagate
- feasible to propagate thanks to:

Metadata

- used in choosing algorithms to execute
- difficult to propagate
- feasible to propagate thanks to:
  - static typing

Metadata

- used in choosing algorithms to execute
- difficult to propagate
- feasible to propagate thanks to:
  - static typing
  - static analysis

Metadata

- used in choosing algorithms to execute
- difficult to propagate
- feasible to propagate thanks to:
  - static typing
  - static analysis
  - reflection

Metadata

- used in choosing algorithms to execute
- difficult to propagate
- feasible to propagate thanks to:
  - static typing
  - static analysis
  - reflection
  - assertions

Types of optimizations

- static

Types of optimizations

- static
- dynamic

Static optimizations

- currently greedy heuristics

Static optimizations

- currently greedy heuristics
- in future maybe cost-based like in databases

Static optimizations

- currently greedy heuristics
- in future maybe cost-based like in databases
- most focused on minimazing network and disc I/O

Static optimizations techniques

- **Pipelining** - multiple operators executed in one process

Static optimizations techniques

- **Pipelining** - multiple operators executed in one process
- **Removing redundancy** - DryadLINQ removes unnecessary hash- and range-partitioning steps

Static optimizations techniques

- **Pipelining** - multiple operators executed in one process
- **Removing redundancy** - DryadLINQ removes unnecessary hash- and range-partitioning steps
- **Eager aggregation** - as re-partitioning is expensive, down-stream aggregations are put in front of partitioning operators

Static optimizations techniques

- **Pipelining** - multiple operators executed in one process
- **Removing redundancy** - DryadLINQ removes unnecessary hash- and range-partitioning steps
- **Eager aggregation** - as re-partitioning is expensive, down-stream aggregations are put in front of partitioning operators
- **I/O reduction** - where possible TCP-pipes and in-memory FIFO channels are used insted of temporary files. Also, data is compressed to reduce network traffic

Dynamic optimizations

- aggregation to reduce I/O

Dynamic optimizations

- aggregation to reduce I/O
- changing locally graph to a tree where possible

Dynamic optimizations

- aggregation to reduce I/O
- changing locally graph to a tree where possible
- clever splitting and spawning of queries (e.g. by sampling a dataset)

Optimizations for `OrderBy`

- dataset `d` is not already range-partitioned

Optimizations for `OrderBy`

- dataset `d` is not already range-partitioned
- its partitions are not either

Optimizations for `OrderBy`

- dataset `d` is not already range-partitioned
- its partitions are not either

We have:

- DS - determinictic sampling

Optimizations for `OrderBy`

- dataset `d` is not already range-partitioned
- its partitions are not either

We have:

- DS - determinictic sampling
- H - histogram

Optimizations for `OrderBy`

- dataset `d` is not already range-partitioned
- its partitions are not either

We have:

- DS - determinictic sampling
- H - histogram
- D - distribution (repartitioning)

Optimizations for `OrderBy`

- dataset `d` is not already range-partitioned
- its partitions are not either

We have:

- DS - determinictic sampling
- H - histogram
- D - distribution (repartitioning)
- M - merging

Optimizations for `OrderBy`

- dataset `d` is not already range-partitioned
- its partitions are not either

We have:

- DS - determinictic sampling
- H - histogram
- D - distribution (repartitioning)
- M - merging
- S - sorting

Optimizations for `OrderBy`

- dataset `d` is not already range-partitioned
- its partitions are not either

We have:

- DS - determinictic sampling
- H - histogram
- D - distribution (repartitioning)
- M - merging
- S - sorting
- S - select many

Optimizations for `OrderBy`

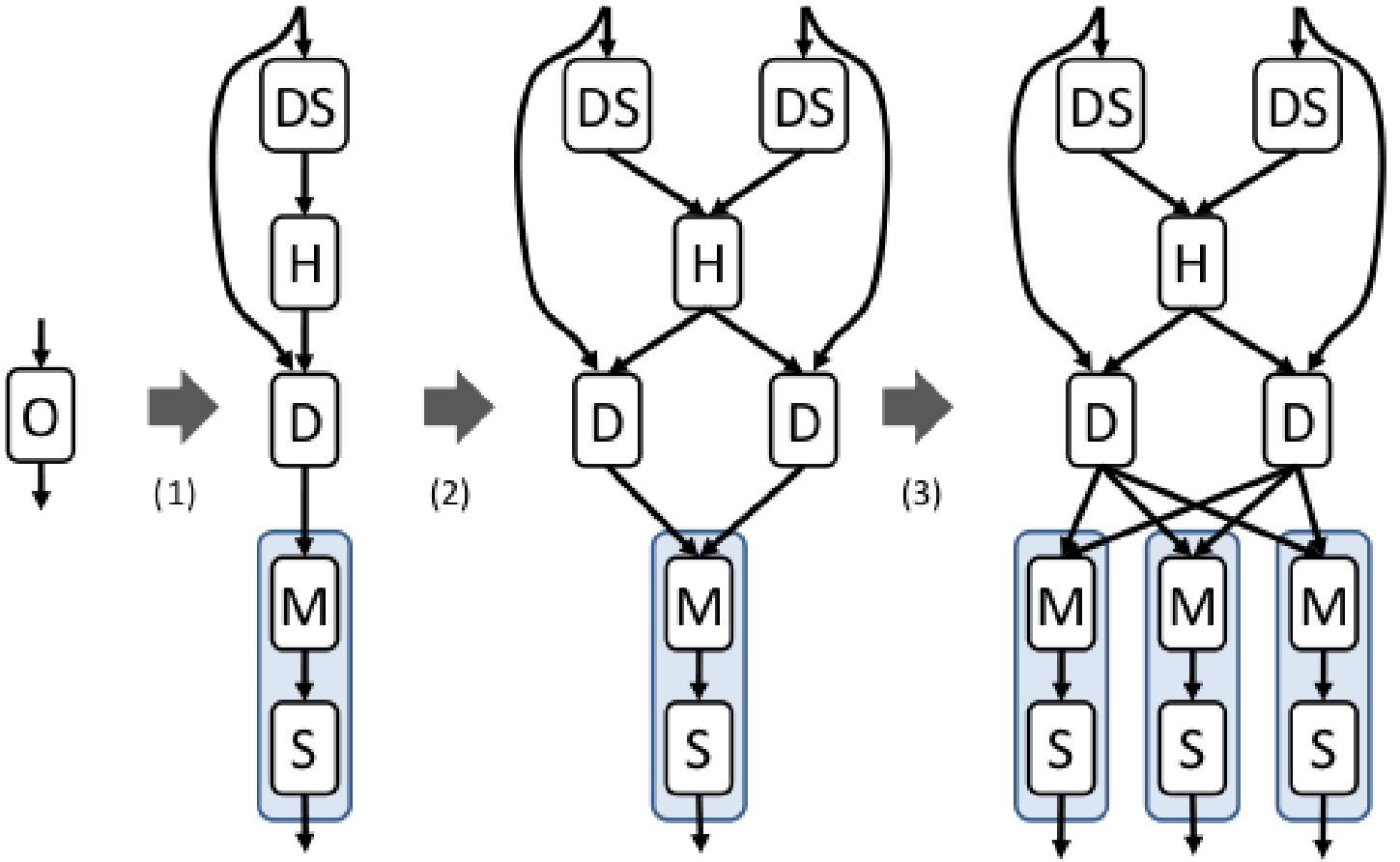

Figure 5: *Distributed sort optimization described in Section 4.2.3. Transformation (1) is static, while (2) and (3) are dynamic.*

Optimizations for MapReduce

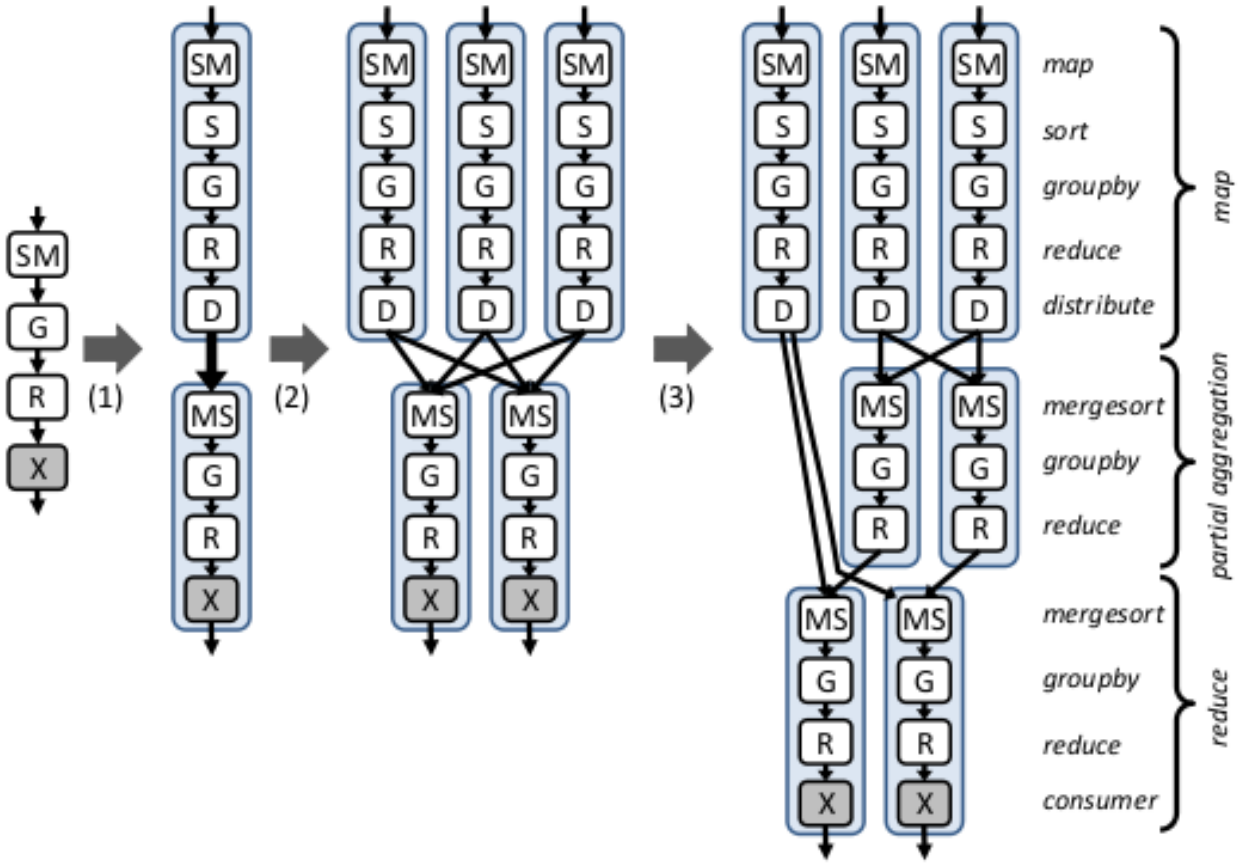

Figure 6: *Execution plan for MapReduce, described in Section 4.2.4. Step (1) is static, (2) and (3) are dynamic based on the volume and location of the data in the inputs.*

What code is generated?

- EPG contains all necessary information but it is not a runnable program

What code is generated?

- EPG contains all necessary information but it is not a runnable program
- DryadLINQ uses dynamic code generation to automatically synthesize LINQ code to be run at the Dryad vertices

What code is generated?

- EPG contains all necessary information but it is not a runnable program
- DryadLINQ uses dynamic code generation to automatically synthesize LINQ code to be run at the Dryad vertices
- the generated code is compiled into a .NET assembly that is shipped to cluster computers at execution time.

What code is generated?

- EPG contains all necessary information but it is not a runnable program
- DryadLINQ uses dynamic code generation to automatically synthesize LINQ code to be run at the Dryad vertices
- the generated code is compiled into a .NET assembly that is shipped to cluster computers at execution time.

For every execution-plan stage, assembly contains two pieces of code:

- the code for the LINQ subexpression executed by each node

What code is generated?

- EPG contains all necessary information but it is not a runnable program
- DryadLINQ uses dynamic code generation to automatically synthesize LINQ code to be run at the Dryad vertices
- the generated code is compiled into a .NET assembly that is shipped to cluster computers at execution time.

For every execution-plan stage, assembly contains two pieces of code:

- the code for the LINQ subexpression executed by each node
- serialization code for the channel data

How do we deal with the local context?

The EPG is created in the original client computer's context and may depend on this context in two ways:

- the expression may reference variables in the local context

How do we deal with the local context?

The EPG is created in the original client computer's context and may depend on this context in two ways:

- the expression may reference variables in the local context
  - variables are computed and we immediately substitute them with their real values

How do we deal with the local context?

The EPG is created in the original client computer's context and may depend on this context in two ways:

- the expression may reference variables in the local context
  - variables are computed and we immediately substitute them with their real values
- the expression may reference .NET libraries

How do we deal with the local context?

The EPG is created in the original client computer's context and may depend on this context in two ways:

- the expression may reference variables in the local context
  - variables are computed and we immediately substitute them with their real values
- the expression may reference .NET libraries
  - objects are evaluated as far as possible and propagated in a serialized form

Why use do we have from other LINQ systems?

Using LINQ allows to leverage other systems using the same constructs:

- PLINQ

Why use do we have from other LINQ systems?

Using LINQ allows to leverage other systems using the same constructs:

- PLINQ
  - is used to run a program on a single machine with multiple processores/cores

Why use do we have from other LINQ systems?

Using LINQ allows to leverage other systems using the same constructs:

- PLINQ
  - is used to run a program on a single machine with multiple processores/cores
  - is called by DryadLINQ when computations are being done on a single machine

Why use do we have from other LINQ systems?

Using LINQ allows to leverage other systems using the same constructs:

- PLINQ
  - is used to run a program on a single machine with multiple processores/cores
  - is called by DryadLINQ when computations are being done on a single machine
- LINQ-to-SQL

Why use do we have from other LINQ systems?

Using LINQ allows to leverage other systems using the same constructs:

- PLINQ
  - is used to run a program on a single machine with multiple processores/cores
  - is called by DryadLINQ when computations are being done on a single machine
- LINQ-to-SQL
  - allows DryadLINQ vertices directly access data in SQL databases

Why use do we have from other LINQ systems?

Using LINQ allows to leverage other systems using the same constructs:

- PLINQ
  - is used to run a program on a single machine with multiple processores/cores
  - is called by DryadLINQ when computations are being done on a single machine
- LINQ-to-SQL
  - allows DryadLINQ vertices directly access data in SQL databases
  - DryadLINQ programs may use partitioned databases as input and output

Why use do we have from other LINQ systems?

Using LINQ allows to leverage other systems using the same constructs:

- PLINQ
  - is used to run a program on a single machine with multiple processores/cores
  - is called by DryadLINQ when computations are being done on a single machine
- LINQ-to-SQL
  - allows DryadLINQ vertices directly access data in SQL databases
  - DryadLINQ programs may use partitioned databases as input and output
- LINQ-to-Objects

Why use do we have from other LINQ systems?

Using LINQ allows to leverage other systems using the same constructs:

- PLINQ
  - is used to run a program on a single machine with multiple processores/cores
  - is called by DryadLINQ when computations are being done on a single machine
- LINQ-to-SQL
  - allows DryadLINQ vertices directly access data in SQL databases
  - DryadLINQ programs may use partitioned databases as input and output
- LINQ-to-Objects
  - allows to run and test DryadLINQ programs on a single computer

Why use do we have from other LINQ systems?

Using LINQ allows to leverage other systems using the same constructs:

- PLINQ
  - is used to run a program on a single machine with multiple processores/cores
  - is called by DryadLINQ when computations are being done on a single machine
- LINQ-to-SQL
  - allows DryadLINQ vertices directly access data in SQL databases
  - DryadLINQ programs may use partitioned databases as input and output
- LINQ-to-Objects
  - allows to run and test DryadLINQ programs on a single computer
  - we can use Visual Studio debugger

Debugging

- checking correctness of the programs is not difficult (thanks to strong typing and narrow interface)

Debugging

- checking correctness of the programs is not difficult (thanks to strong typing and narrow interface)
- performance debugging is problematic

Debugging

- checking correctness of the programs is not difficult (thanks to strong typing and narrow interface)
- performance debugging is problematic
- in future there will be tools to perform effective log-mining

Hardware used in experiments

- cluster of 240 computers

Hardware used in experiments

- cluster of 240 computers
- Windows Server 2003 64-bit operating system

Hardware used in experiments

- cluster of 240 computers
- Windows Server 2003 64-bit operating system
- dual-core AMD Operon 2218 HE CPU (2.6GHz)

Hardware used in experiments

- cluster of 240 computers
- Windows Server 2003 64-bit operating system
- dual-core AMD Operon 2218 HE CPU (2.6GHz)
- 16GB DDR2 RAM

Hardware used in experiments

- cluster of 240 computers
- Windows Server 2003 64-bit operating system
- dual-core AMD Operon 2218 HE CPU (2.6GHz)
- 16GB DDR2 RAM
- 750GB SATA hard drive

Hardware used in experiments

- cluster of 240 computers
- Windows Server 2003 64-bit operating system
- dual-core AMD Operon 2218 HE CPU (2.6GHz)
- 16GB DDR2 RAM
- 750GB SATA hard drive
- computers connected to a Linksys SRW2048 48-port full-crossbar GBit Ethernet local switch via GBit Ethernet

Hardware used in experiments

- cluster of 240 computers
- Windows Server 2003 64-bit operating system
- dual-core AMD Operon 2218 HE CPU (2.6GHz)
- 16GB DDR2 RAM
- 750GB SATA hard drive
- computers connected to a Linksys SRW2048 48-port full-crossbar GBit Ethernet local switch via GBit Ethernet
- 29 to 31 computers connected to each switch

Hardware used in experiments

- cluster of 240 computers
- Windows Server 2003 64-bit operating system
- dual-core AMD Operon 2218 HE CPU (2.6GHz)
- 16GB DDR2 RAM
- 750GB SATA hard drive
- computers connected to a Linksys SRW2048 48-port full-crossbar GBit Ethernet local switch via GBit Ethernet
- 29 to 31 computers connected to each switch
- each local switch connected to a central Linksys2048 switch via 6 ports aggregated using 802.3ad link aggregation

Terasort

Terasort benchmark

The task is to sort 10 billion 100-Byte records using case-insensitive string comparison on a 10-Byte key

Assumptions and facts

- each computer in the cluster stored a partition of size around 3.87GB

Assumptions and facts

- each computer in the cluster stored a partition of size around 3.87GB
- number of computers vary, so total size of data is $3.87n$GB

Assumptions and facts

- each computer in the cluster stored a partition of size around 3.87GB
- number of computers vary, so total size of data is $3.87n$GB
- on the largest run $n = 240$ and we sort 10^12 bytes of data

Assumptions and facts

- each computer in the cluster stored a partition of size around 3.87GB
- number of computers vary, so total size of data is $3.87n$GB
- on the largest run $n = 240$ and we sort 10^12 bytes of data
- the most time-consuming phase is the network read and range-partitioning the data

Assumptions and facts

- each computer in the cluster stored a partition of size around 3.87GB
- number of computers vary, so total size of data is $3.87n$GB
- on the largest run $n = 240$ and we sort 10^12 bytes of data
- the most time-consuming phase is the network read and range-partitioning the data

Computers	1	2	10	20	40	80	240
Time	119	241	242	245	271	294	319

Table 1: Time in seconds to sort different amounts of data. The total data sorted by an n-machine experiment is around $3.87n$GBytes, or 10^{12} Bytes when $n = 240$.

Results

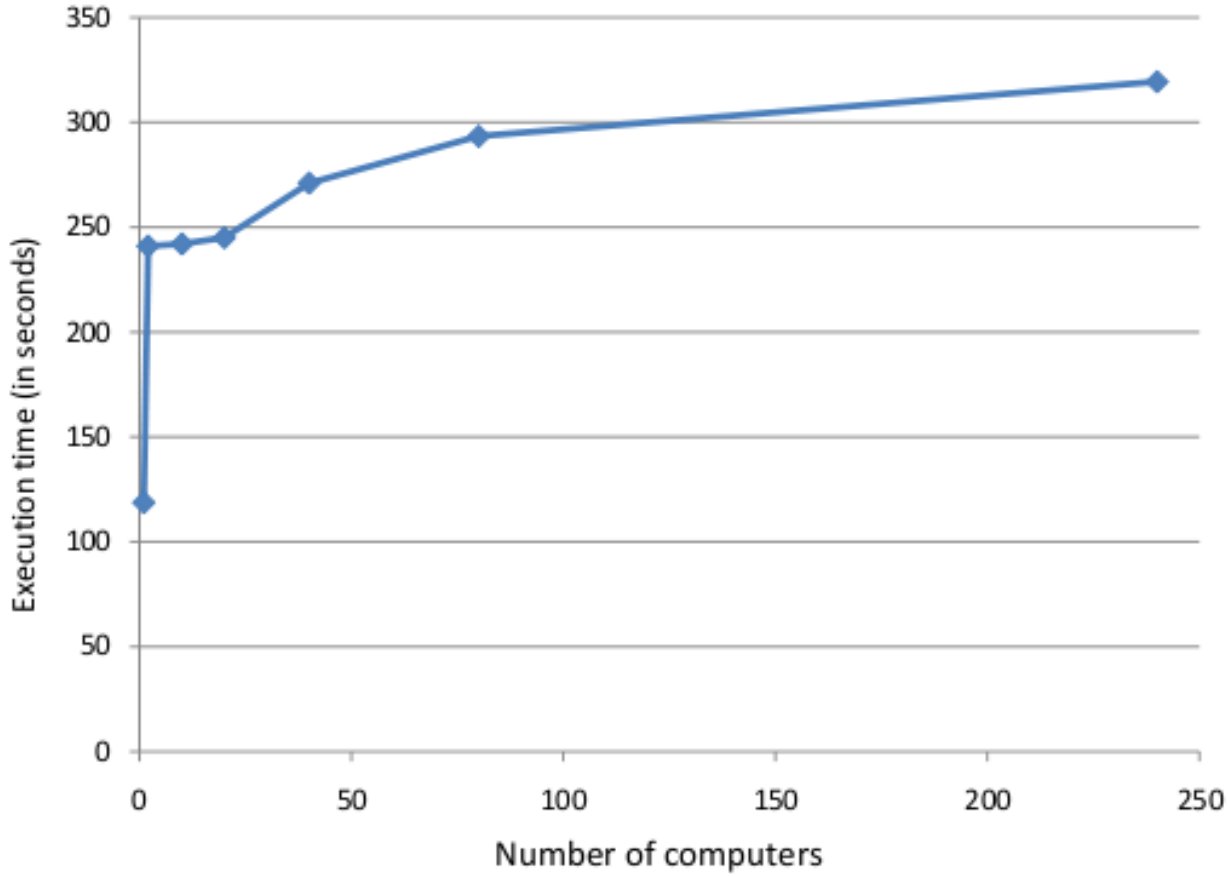

Figure 7: *Sorting increasing amounts of data while keeping the volume of data per computer fixed. The total data sorted by an n-machine experiment is around $3.87n$ GBytes, or 10^{12} Bytes when $n = 240$.*

SkyServer

SkyServer

There has been implemented the most time-consuming query (Q18) from the Sloan Digital Sky Survey database. It uses a three-way Join over two input tables containing 11.8 GBytes and 41.8 GBytes of data, respectively.

In this experiment, the performance of the two-pass variant of the Dryad program described in with that of DryadLINQ is compared.

Assumptions and facts

- Dryad program in C++ has over 1000 lines

Assumptions and facts

- Dryad program in C++ has over 1000 lines
- DryadLINQ program in C# language has 100 lines

Assumptions and facts

- Dryad program in C++ has over 1000 lines
- DryadLINQ program in C# language has 100 lines
- input tables were manually partitioned into 40 pieces using the same keys

Assumptions and facts

- Dryad program in C++ has over 1000 lines
- DryadLINQ program in C# language has 100 lines
- input tables were manually partitioned into 40 pieces using the same keys
- we ensure that each computer has approximately $\frac{40}{n}$ partitions to compute

Assumptions and facts

- Dryad program in C++ has over 1000 lines
- DryadLINQ program in C# language has 100 lines
- input tables were manually partitioned into 40 pieces using the same keys
- we ensure that each computer has approximately $\frac{40}{n}$ partitions to compute

Computers	1	5	10	20	40
Dryad	2167	451	242	135	92
DryadLINQ	2666	580	328	176	113

Table 2: Time in seconds to process skyserver Q18 using different number of computers.

Results

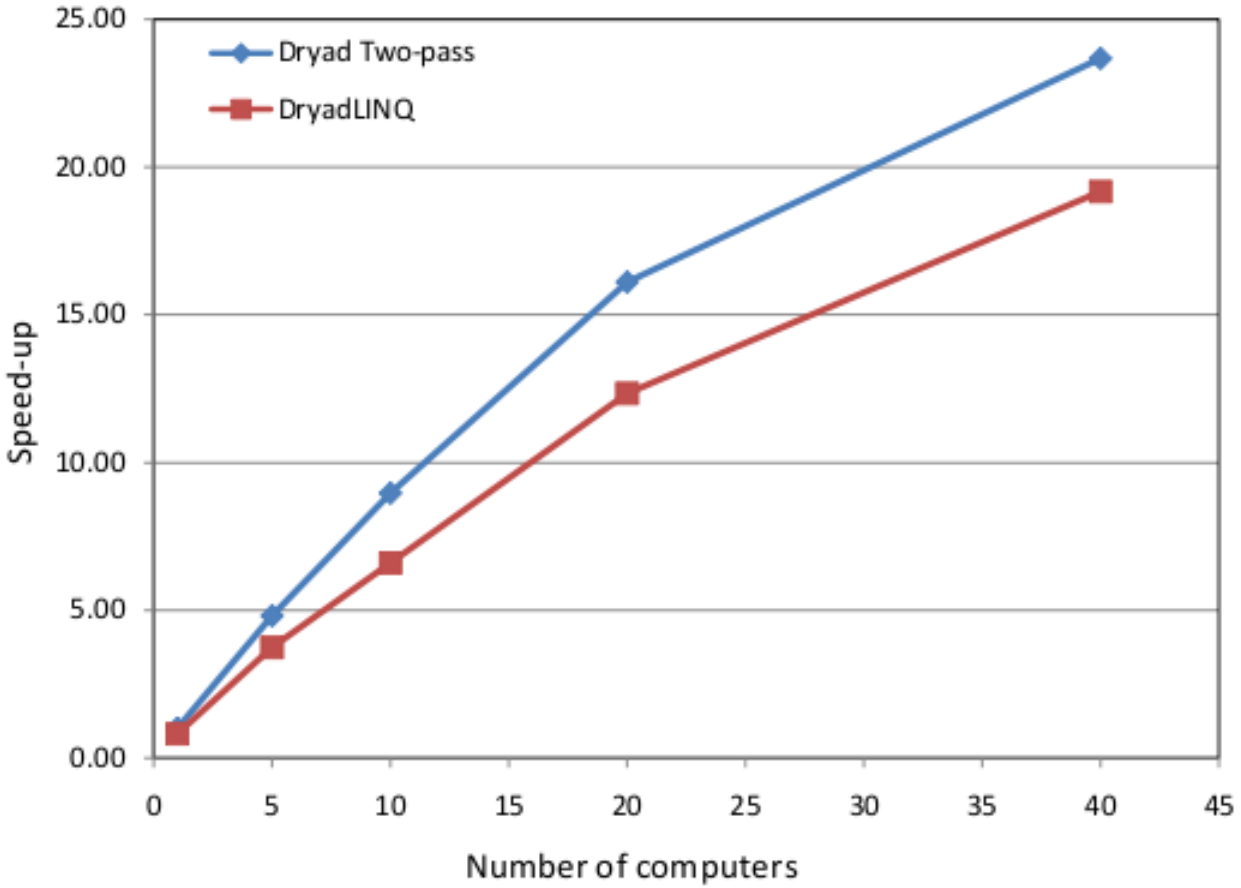

Figure 8: *The speed-up of the Skyserver Q18 computation as the number of computers is varied. The baseline is relative to DryadLINQ job running on a single computer and times are given in Table 2.*

Conclusions

- DryadLINQ was engineered for batch applications on large datasets

Conclusions

- DryadLINQ was engineered for batch applications on large datasets
- it is not suitable for some uses (e.g. where low latency is necessary)

Conclusions

- DryadLINQ was engineered for batch applications on large datasets
- it is not suitable for some uses (e.g. where low latency is necessary)
- it is suitable for problems easily transforming to queries

Conclusions

- DryadLINQ was engineered for batch applications on large datasets
- it is not suitable for some uses (e.g. where low latency is necessary)
- it is suitable for problems easily transforming to queries
- `Apply` operator is overused by inexperienced users

Conclusions

- DryadLINQ was engineered for batch applications on large datasets
- it is not suitable for some uses (e.g. where low latency is necessary)
- it is suitable for problems easily transforming to queries
- `Apply` operator is overused by inexperienced users
- `Apply` operator may speed aplication up a lot

Conclusions

- DryadLINQ was engineered for batch applications on large datasets
- it is not suitable for some uses (e.g. where low latency is necessary)
- it is suitable for problems easily transforming to queries
- `Apply` operator is overused by inexperienced users
- `Apply` operator may speed aplication up a lot
- current research is on optimizer being better in common uses