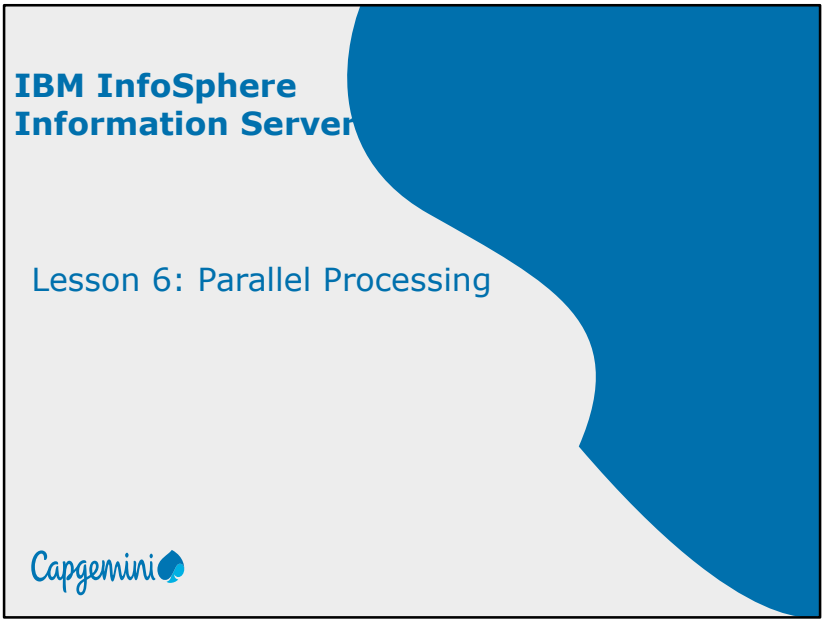

IBM InfoSphere
Information Server
Lesson 6: Parallel Processing
Capgemini

Lesson Objectives

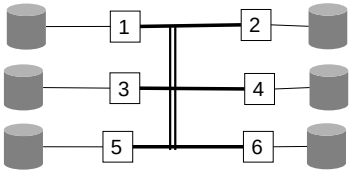
- After completing this unit, you should be able to:
- Describe parallel processing architecture
- Describe pipeline parallelism
- Describe partition parallelism
- List and describe partitioning and collecting algorithms
- Describe configuration files
- Describe the parallel job compilation process
- Explain OSH
- Explain the Score

IBM InfoSphere Information Server

6.1 : Datastage
parallelism and types

Key Parallel Job Concepts

- Parallel processing:
  - Executing the job on multiple CPUs
- Scalable processing:
  - Add more resources (CPUs and disks) to increase system performance



- Example system: 6 CPUs (processing nodes) and disks
- Scale up by adding more CPUs
- Add CPUs as individual nodes or to an SMP system

Parallel processing is the key to building jobs that are highly scalable. The parallel engine uses the processing node concept. A processing node is a CPU or an SMP, or a board on an MPP.

Parallel jobs brings the power of parallel processing to your data extraction and transformation applications.

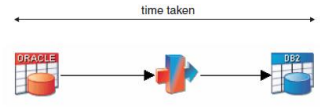
DataStage parallelism

- There are two basic types of parallel processing; pipeline and partitioning. InfoSphere DataStage allows you to use both of these methods
- Pipeline parallelism.:- If you ran the example job on a system with at least three processors, the stage reading would start on one processor and start filling a pipeline with the data it had read.
- The transformer stage would start running on another processor as soon as there was data in the pipeline, process it and start filling another pipeline.

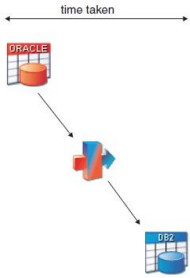
DataStage parallelism

- The stage writing the transformed data to the target database would similarly start writing as soon as there was data available. Thus all three stages are operating simultaneously.
- If you were running sequentially, there would only be one instance of each stage. If you were running in parallel, there would be as many instances as you had partitions

Pipeline Parallelism

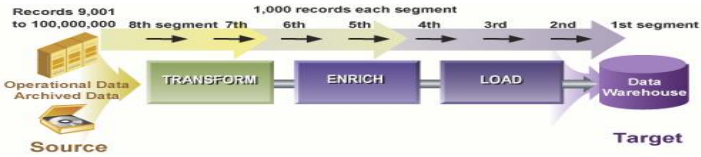


conceptual representation of job running with no parallelism



conceptual representation of same job using pipeline parallelism

Pipeline Parallelism



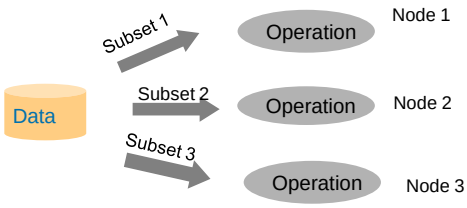
- Transform, clean, load processes execute simultaneously
- Like a conveyor belt moving rows from process to process
  - Start downstream process while upstream process is running
- Advantages:
  - Reduces disk usage for staging areas
  - Keeps processors busy
- Still has limits on scalability

Partition Parallelism

- Divide the incoming stream of data into subsets to be separately processed by an operation
  - Subsets are called partitions (nodes)
- Each partition of data is processed by the same operation
  - E.g., if operation is Filter, each partition will be filtered inexactly the same way
- Facilitates near-linear scalability
  - 8 times faster on 8 processors
  - 24 times faster on 24 processors
  - This assumes the data is evenly distributed

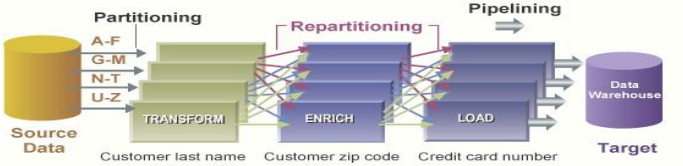
Partitioning breaks a dataset into smaller sets. This is a key to scalability. However, the data needs to be evenly distributed across the partitions; otherwise, the benefits of partitioning are reduced. It is important to note that what is done to each partition of data is the same. How the data is processed or transformed is the same.

Three-Node Partitioning



- Here the data is partitioned into three partitions
- The operation is performed on each partition of data separately and in parallel
- If the data is evenly distributed, the data will be processed three times faster

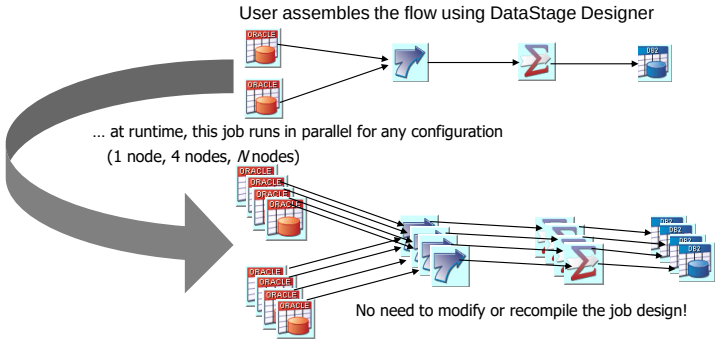
Parallel Jobs Combine Partitioning and Pipelining



- Within parallel jobs pipelining, partitioning, and repartitioning are automatic
- Job developer only identifies:
  - Sequential vs. parallel mode (by stage)
    - Default for most stages is parallel mode
  - Method of data partitioning
    - The method to use to distribute the data into the available nodes
  - Method of data collecting
    - The method to use to collect the data from the nodes into a single node
  - Configuration file
    - Specifies the nodes and resources

By combining both pipelining and partitioning DataStage creates jobs with higher volume through put.
The configuration file drives the parallelism by specifying the number of partitions

Job Design v. Execution



- Much of the parallel processing paradigm is hidden from the programmer. The programmer simply designates the process flow, as shown in the upper portion of this diagram.
- The parallel engine, using the definitions in the configuration file, will actually execute processes that are partitioned and parallelized, as illustrated in the bottom portion.

IBM InfoSphere Information Server

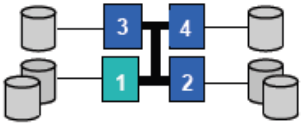
6.2 : Configuration File

Configuration File

- Configuration file separates configuration (hardware / software) from job design
  - Specified per job at runtime by $APT_CONFIG_FILE
  - Change hardware and resources without changing job design
- Defines nodes with their resources (need not match physical CPUs)
  - Dataset, Scratch, Buffer disk (file systems)
  - Optional resources (Database, SAS, etc.)
  - Advanced resource optimizations
    - "Pools" (named subsets of nodes)
- Multiple configuration files can be used different occasions of job execution
  - Optimizes overall throughput and matches job characteristics to overall hardware resources
  - Allows runtime constraints on resource usage on a per job basis

DataStage jobs can point to different configuration files by using job parameters. Thus, a job can utilize different hardware architectures without being recompiled. It can pay to have a 4-node configuration file running on a 2 processor box, for example, if the job is "resource bound." We can spread disk I/O among more controllers.

Example Configuration File



- Key points:

1. Number of nodes defined.
2. Resources assigned to each node. Their order is significant .
3. Advanced resource optimizations and configuration (named pools, database, SAS).

```
{
node " n1 " {
fastname " s1 "
pool " " " n 1" " s 1" "app2 " " sor t"
resource disk " /orch / n1 / d1 " { }
resource disk " / orch / n1 / d2 " { "
    bigdata " }
resource scratchdisk "/ temp " {" s or
    t "}
}
node " n2" {
fastname " s2 "
pool " " " n 2" " s 2" "app1 "
resource disk " /orch / n2 / d1 " { }
resource disk " / orch / n2 / d2 " { "
    bigdata " }
resource scratchdisk "/ temp " {}
}
node " n3" {
fastname " s3 "
pool " " " n 3" " s 3" "app1 "
resource disk " /orch / n3 / d1 " { }
resource scratchdisk "/ temp " {}
}
node " n4" {
fastname " s4 "
pool " " " n 4" " s 4" "app1 "
resource disk " /orch / n4 / d1 " { }
resource scratchdisk "/ temp " {}
}
}
```

IBM InfoSphere Information Server

6.2 : Partioning and Collecting

Partitioning and Collecting

- Partitioning breaks incoming rows into multiple streams of rows (one for each node)
- Each partition of rows is processed separately by the stage/operator
- Collecting returns partitioned data back to a single stream
- Partitioning / Collecting is specified on stage input links

Partitioning / Collecting Algorithms

- Partitioning algorithms include:
  - Round robin
  - Random
  - Hash: Determine partition based on key value
    - Requires key specification
  - Modulus
  - Entire: Send all rows down all partitions
  - Same: Preserve the same partitioning
  - Auto: Let DataStage choose the algorithm

Partitioning / Collecting Algorithms

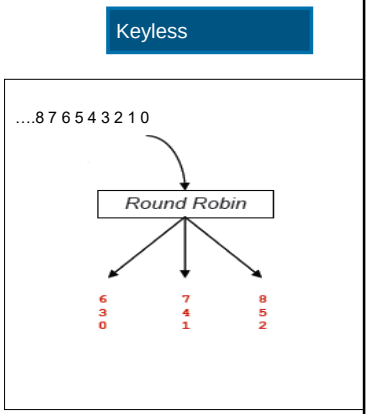
- Collecting algorithms include:
  - Round robin
  - Auto
    - Collect first available record
  - Sort Merge
    - Read in by key
    - Presumes data is sorted by the key in each partition
    - Builds a single sorted stream based on the key
  - Ordered
    - Read all records from first partition, then second, …

Keyless V. Keyed Partitioning Algorithms

- Keyless: Rows are distributed independently of data values
  - Round Robin
  - Random
  - Entire
  - Same
- Keyed: Rows are distributed based on values in the specified key
  - Hash: Partition based on key
    - Example: Key is State. All "CA" rows go into the same partition; all "MA" rows go in the same partition. Two rows of the same state never go into different partitions
  - Modulus: Partition based on modulus of key divided by the number of partitions. Key is a numeric type.
    - Example: Key is Order Number (numeric type). Rows with the same order number will all go into the same partition.
  - DB2: Matches DB2 EEE partitioning

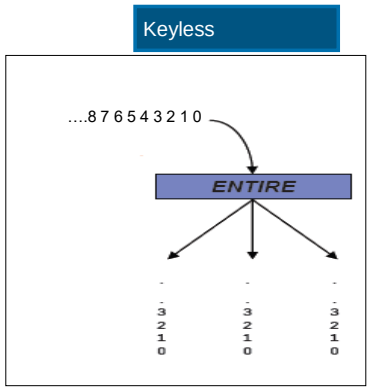
Round Robin and Random Partitioning

- Keyless partitioning methods
- Rows are evenly distributed across partitions
  - Good for initial import of data if no other partitioning is needed
  - Useful for redistributing data
- Fairly low overhead
- Round Robin assigns rows to partitions like dealing cards
  - Row/Partition assignment will be the same for a given $APT_CONFIG_FILE
- Random has slightly higher overhead, but assigns rows in anon-deterministic fashion between job runs



ENTIRE Partitioning

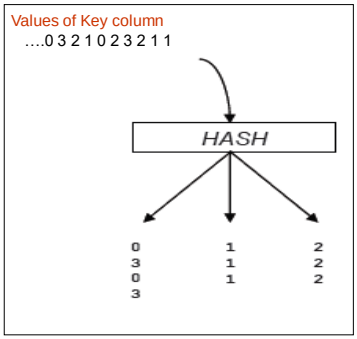
- Each partition gets a complete copy of the data
  - –Useful for distributing **lookup** and reference data
    - May have performance impact in MPP / clustered environments
  - On SMP platforms, Lookup stage (only) uses shared memory instead of duplicating ENTIRE reference data
    - On MPP platforms, each server uses shared memory for a single local copy
- ENTIRE is the default partitioning for Lookup reference links with "Auto" partitioning
  - On SMP platforms, it is a good practice to set this explicitly on the link(s) *Normal* Lookup reference



HASH Partitioning

- Keyed partitioning method
- Rows are distributed according to the values in key columns
  - Guarantees that rows with same key values go into the same partition
  - Needed to prevent matching rows from "hiding" in other partitions
    - E.g. Join, Merge, Remove Duplicates,…
  - Partition distribution is relatively equal if the data across the source column(s) is evenly key distributed

Keyed



HASH Partitioning

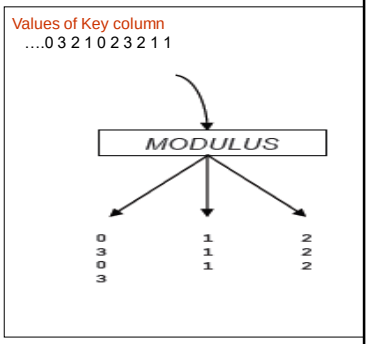
- For certain stages (Remove Duplicates, Join, Merge) to work correctly in parallel, the user must override the default (Auto) to Hash or a variant: Range, Modulus.
- Here the numbers are no longer row IDs, but values of key column.
  - Hash guarantees that all the rows with key value 3 end up in the same partition.
  - Hash does not guarantee "continuity": here, 3s are bunched with 0s, not with neighboring value 2.
    - This is an expensive version of Hash, "Range," that guarantees continuity.
  - Hash does not guarantee load balance.
- Make sure key column(s) takes on enough values to distribute data across available partitions.

  ("gender" would be a poor choice of key…)

Modulus Partitioning

- Keyed partitioning method
- Rows are distributed according to the values in one integer key column
  - Uses modulus
    partition = MOD (key_value / #partitions)
- Faster than HASH
- Guarantees that rows with identical key values go in the same partition
- Partition size is relatively equal if the data within the key column is evenly distributed

Keyed



Auto Partitioning

- DataStage inserts partition components as necessary to ensure correct results
  - Before any stage with "Auto" partitioning
  - Generally chooses *ROUND-ROBIN* or *SAME*
    - chooses Round Robin when going from sequential to parallel.
    - chooses Same when going from parallel to parallel.
  - Inserts *HASH* on stages that require matched key values (e.g. Join, Merge, Remove Duplicates)
  - Inserts ENTIRE on Normal (not Sparse) Lookup reference links
    - NOT always appropriate for MPP/clusters

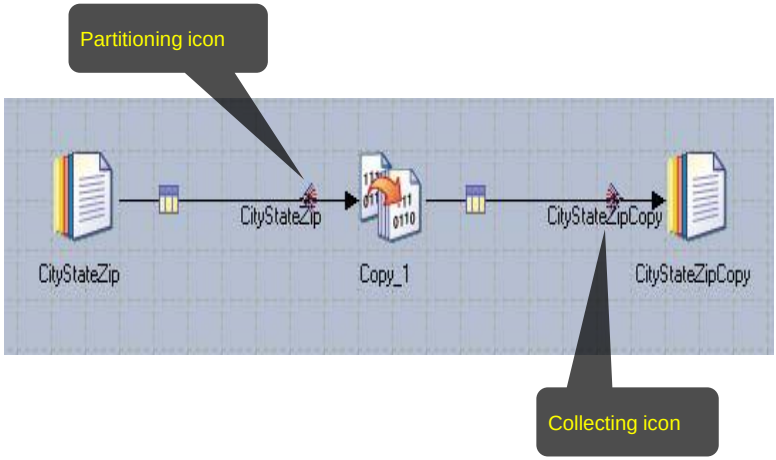
Auto Partitioning

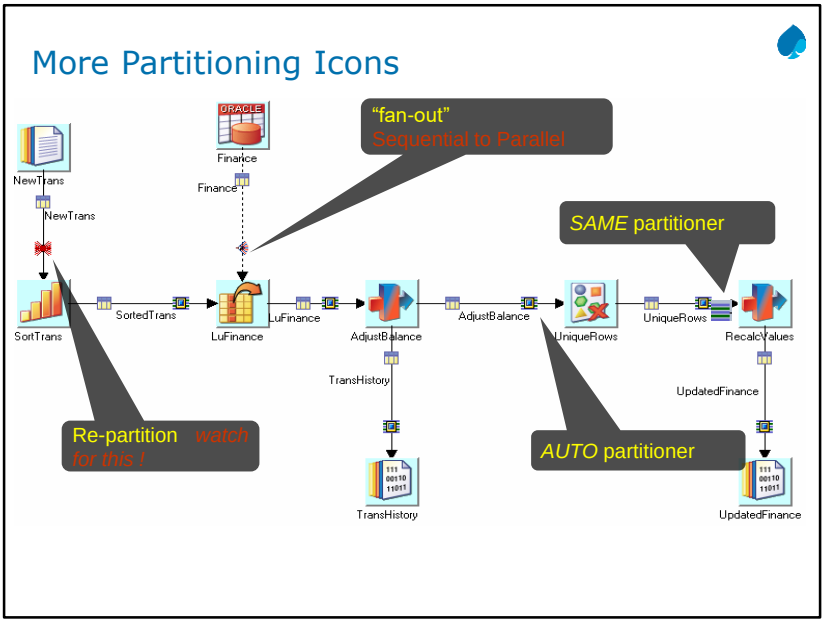
- Since DataStage has limited awareness of your data and business rules, explicitly specify HASH partitioning
  - when needed when processing requires groups of related records.
- DataStage has no visibility into Transformer logic
- Hash is required before Sort and Aggregator stages
- DataStage sometimes inserts un-needed partitioning
  - Check the log

Partitioning Requirements for Related Records

- Misplaced records
  - Using Aggregator stage to sum customer sales by customer number
  - If there are 25 customers, 25 records should be output
  - But suppose records with the same customer numbers are spread across partitions
    - This will produce more than 25 groups (records)
  - Solution: Use hash partitioning algorithm
- Partition imbalances
  - If all the records are going down only one of the nodes, then the job is in effect running sequentially

Partitioning / Collecting Link Icons




More Partitioning Icons
"fan-out"
Sequential to Parallel
SAME partitioner
Re-partition watch for this !
AUTO partitioner
NewTrans
Finance
SortTrans
SortedTrans
LuFinance
AdjustBalance
UniqueRows
RecalcValues
TransHistory
UpdatedFinance

More Partitioning Icons

- Partitioners and collectors have no stage nor icons of their own.
- They live on input links of stages running in parallel / sequentially, respectively.
  - Link markings indicate their presence.
  - S----------->S (no Marking)
  - S----(fan out)--->P (partitioner)
  - P----(fan in) ---->S (collector)
  - P----(box)------->P (no reshuffling: partitioner using "SAME" method)
  - P----(bow tie)--->P (reshuffling: partitioner using another method)

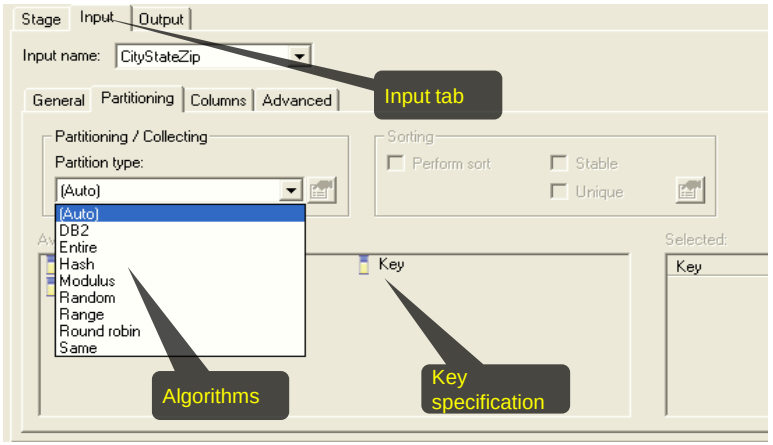
More Partitioning Icons

- Collectors = inverse partitioners
- recollect rows from partitions into a single input stream to a sequential stage

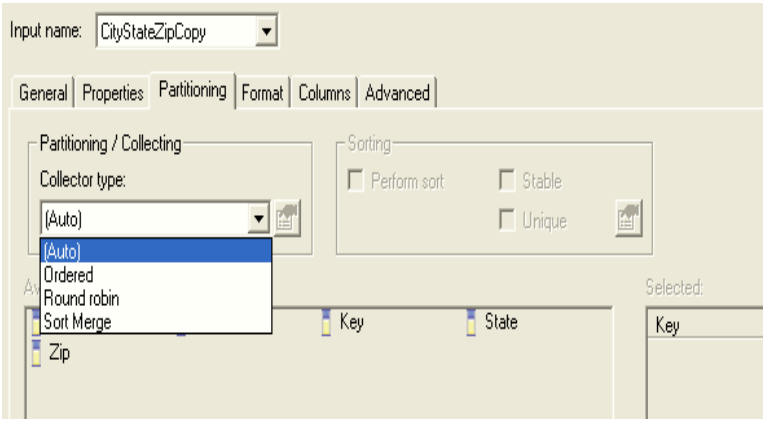
They are responsible for some surprising behavior:

- The default (Auto) is "eager" to output rows and typically causes non-determinism :row order may vary from run to run with identical input.
- In the example of repartitioning, the New Trans file is using multiple readers, so the data is being read into multiple partitions. Therefore, a reshuffling occurs at the Sort stage.

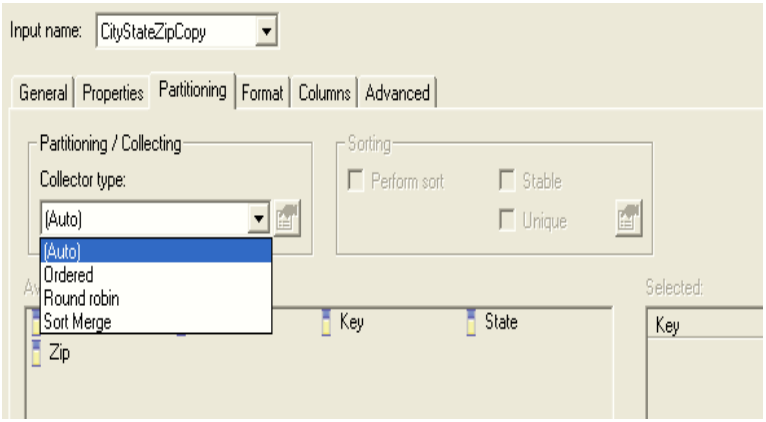
Partitioning Tab



Collecting Specification



Collecting Specification



Q&A

- What file defines the degree of parallelism a job runs under?
- Name two partitioning algorithms that partition based on key values?
- What partitioning algorithm produces the most even distribution of data in the various partitions?

Q&A

- Configuration file
- Hash, Modulus, DB2
- Round robin. Or Entire