independIT Integrative Technologies GmbH
Bergstraße 6
D–86529 Schrobenhausen

schedulix!focus

Workflow Parallelisation using the schedulix Dynamic Submit Function

Dieter Stubler Ronald Jeninga

November 25, 2016

Workflow Parallelisation using the schedulix Dynamic Submit Function

Introduction

Processing large volumes of data is a typical feature of data warehouse environments. Depending on the available hardware resources, sooner or later there comes a point where a processing step can no longer be reproduced by being run on a processor or as a process. There are different reasons for this:

- Time-related requirements require the use of multiple processors.
- System resources (memory, disk space, temporary tablespace, rollback segments, . . .) are limited in their availability.
- Any errors that occur require frequent repetitions of the processing steps.

Parallelisation with RDBMS Parallel Processing

Contemporary database systems allow parallel query processing. This feature allows queries, and in some cases manipulations of large volumes of data, to be parallelised internally in the database and run on multiple processors.

The advantages of this solution are:

- No or minimal development requirements.
- Low overheads due to the parallelisation.

The disadvantages, however, are:

- Controlling the degree of parallelism is only restricted or is time-consuming.
- Adapting the number of parallel sub-processes to be executed at any one time during runtime is generally not possible.
- When an error occurs, the work that has already been done is lost completely.
- The requisite database system resources (temporary tablespace, rollback segments, . . .) have to be sufficiently available for the entire operation.
- In systems where control over the resources is crucial, processes that are parallelised internally in a database are problematic because their resource consumption can often not be determined.
- The impact of parallelisation on the rest of the system is difficult to predict or plan.

RDBMS Parallel Processing is therefore primarily suitable for accelerating operations through the use of multiple processors for one processing step. If system resources are not
bountifully available, the disadvantages will become even more tangible. This applies especially, despite parallelisation, to long-running processing steps.

Parallelisation at application level

As an alternative to RDBMS internal parallelisation, it is usually possible to break down a processing step on the application side into parallel executable sub-tasks and then run them concurrently.

The advantages of this solution are:

- Full control of the degree of parallelism is possible.
- The number of active parallel processes at any one time can be dynamically controlled.
- Errors in one parallel process do not invalidate the work of other, successfully completed parallel processes. This diminishes the effects of errors on the total runtime.
- System resources only have to be made available for the concurrently running parallel processes.
- The resource consumption can be better planned and the impact on the rest of the system can be dynamically influenced.

The disadvantages, however, are:

- Implementation is extremely complex without the support of a suitable scheduling system.
- The overheads for combining the results are usually higher than is the case with RDBMS internal parallelisation.

This comparison between the advantages and disadvantages of application-based parallelisation clearly demonstrates that, especially where extremely complex and long-running processing procedures in environments with a limited availability of resources are concerned, application-based parallelisation is a much more viable solution than RDBMS internal parallelisation. Processing scenarios with the aforementioned characteristics are commonplace in data warehouse environments.

Implementation

The implementation of application-based parallelisation always requires the following 3 (4) sub-tasks:

1. Decomposition of a processing step into parallel executable sub-processes.
2. Realisation of the sub-process.
3. Control over the parallel execution of the sub-processes.
4. Optionally, combining of the part results into one overall result.

Example:

In a data warehouse there is an SQL script which accesses an extremely large, partitioned database table to create an aggregation and then saves it in a results table. Since the processing operation meanwhile surpasses the available TEMP capacity of the database, it needs to be parallelised.
Realising the first step essentially involves creating a list of the partitions and then triggering a sub-process for each of these partitions.
For the second step we use the original SQL script. We modify this so that the aggregation does not take place using the entire table, but just one partition that is handed over as a parameter. We then save the result in an intermediate aggregate.
After all the parallel sub-processes have been completed, for the fourth step the intermediate aggregate has to be aggregated again and written to the results table. The original SQL script provides a good template for this task as well.
These realisation measures can usually be implemented in just a few hours.
The problem child in the realisation is step 3.
To benefit from all the advantages of application-based parallelisation, it is necessary to implement a control mechanism which provides at least the following functions:

- Execution and error validation of the parallel sub-processes.
- Ability to control the number of concurrently running parallel processes (at runtime).
- Monitoring and restarting of sub-processes after an error.

If a custom-built solution has to be developed here (scripting, . . .), this will require considerable know-how and it will be scarcely possible to achieve an efficient, stable and also maintainable solution at a justifiable cost. If the parallel sub-processes cannot be integrated in the deployed scheduling system, they cannot be effectively controlled and steered during operation of the overall data warehouse system.
The schedulix Scheduling System, on the other hand, has all the features required to completely cover step 3 of the implementation (see above). This means that

no time and costs whatsoever have to be invested in the development in order to realise even the most complicated task in this context. The greatest obstacle to process parallelisation at application level is thus eliminated.

Implementation in the schedulix Scheduling System with Dynamic Submits

The schedulix Scheduling System allows child jobs that are hierarchically subordinate to a parent job to be created using different parameters via the API schedulix. These are then visible in the scheduling system just like all the other jobs in a workflow. All the functions (monitoring, operation, resource management, ...) in the schedulix Scheduling System are unreservedly available for these dynamically created job instances as well.

To go back to the example:

Figure 1 shows the definition of the parallelised workflow in the schedulix Scheduling System.

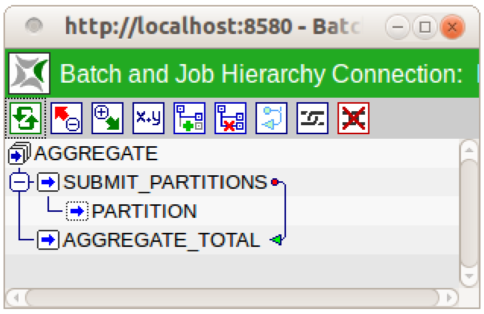

http://localhost:8580 - Batc
Batch and Job Hierarchy Connection:
AGGREGATE
SUBMIT_PARTITIONS
PARTITION
AGGREGATE_TOTAL

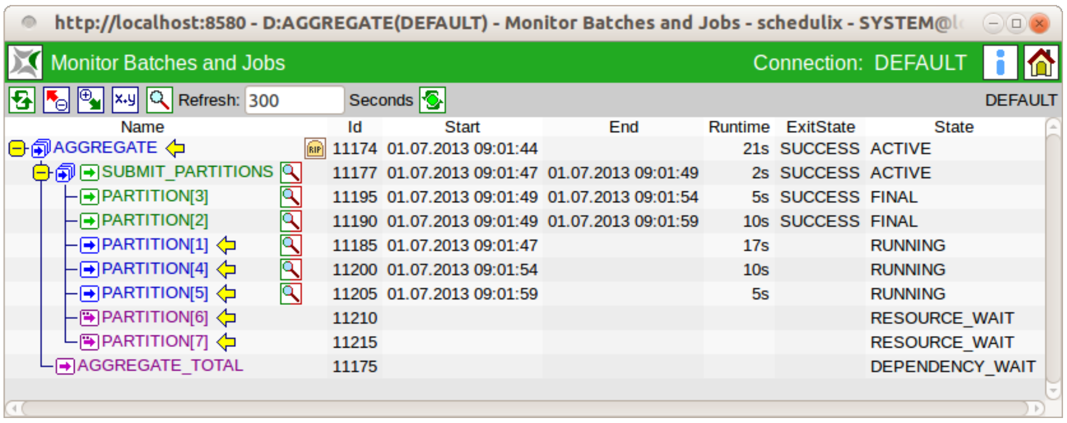
Figure 1: Definition of the aggregation in the schedulix Scheduling System

Starting the AGGREGATE batch with a submit creates the two static child jobs SUBMIT_PARTITIONS and AGGREGATE_TOTAL and the SUBMIT_PARTITIONS job can be started. Due to its dependency (arrow in the screenshot) on SUBMIT_PARTITIONS, AGGREGATE_TOTAL is only run after the SUBMIT_PARTITIONS job (including all its child jobs) has been completed.
The underlying program for the SUBMIT_PARTITIONS job is the realisation of step 1 of our sub-tasks for the parallelisation. It identifies the partitions in the database table and creates a child job for each partition via the schedulix API (e.g.: command line command 'submit').
The command to be executed for the SUBMIT_PARTITIONS job can be defined in a UNIX environment as follows:

```
sh -c "for P in P1 P2 P3 P4 P5 P6 P7
do
```

```
        submit --host $SDMSHOST --port $SDMSPORT --jid $JOBID \\
            --key $KEY --job PARTITION --tag \$P PARTITION \$P
        if [ \$? != 0]; then exit 1; fi
    done"
```

A running AGGREGATE batch is displayed in the schedulix monitoring module as shown in Figure 2.


http://localhost:8580 - D:AGGREGATE(DEFAULT) - Monitor Batches and Jobs - schedulix - SYSTEM@l
Monitor Batches and Jobs
Connection: DEFAULT
Refresh: 300
Seconds
DEFAULT

Name	Id	Start	End	Runtime	ExitState	State
AGGREGATE	11174	01.07.2013 09:01:44		21s	SUCCESS	ACTIVE
SUBMIT_PARTITIONS	11177	01.07.2013 09:01:47	01.07.2013 09:01:49	2s	SUCCESS	ACTIVE
PARTITION[3]	11195	01.07.2013 09:01:49	01.07.2013 09:01:54	5s	SUCCESS	FINAL
PARTITION[2]	11190	01.07.2013 09:01:49	01.07.2013 09:01:59	10s	SUCCESS	FINAL
PARTITION[1]	11185	01.07.2013 09:01:47		17s		RUNNING
PARTITION[4]	11200	01.07.2013 09:01:54		10s		RUNNING
PARTITION[5]	11205	01.07.2013 09:01:59		5s		RUNNING
PARTITION[6]	11210					RESOURCE_WAIT
PARTITION[7]	11215					RESOURCE_WAIT
AGGREGATE_TOTAL	11175					DEPENDENCY_WAIT

Figure 2: Monitoring window showing an active aggregation

Every single one of the AGGREGATE_PARTITION jobs can now be monitored via the schedulix Scheduling System and individually repeated (rerun) in the event of an error.
In just a few steps, by defining a resource and the requirements for this resource the AGGREGATE_PARTITION job can be used to control how many of the AGGREGATE_PARTITION jobs are to be run by the system at any one time. This number can also be changed during runtime. Figure 2 shows the workflow set to run a maximum of three parallel AGGREGATE_PARTITION jobs. In addition, the AGGREGATE_PARTITION jobs can be integrated into the resource management of the remaining data warehouse operation by means of requirements for resources for mapping the available system resources.

Closing remarks

With its Dynamic Submit functionality, the schedulix Scheduling System allows a fast, cost-effective, stable and maintainable implementation of application-based parallelisation.
The parallel sub-processes are integrated into the workflow and are thus visualised in the scheduling system, ensuring an overview and control over each of these sub-processes at all times.