Robust Execution of massive compute-intensive calculations with SEE

Holger Wenzel[1], Erich Schelkle[2]

[1]Dassault Systemes SIMULIA SLM, [2]Automotive Simulation Center Stuttgart

Abstract: *Using the SIMULIA Execution Engine (SEE) to perform multi-run analyzes like Design-of-Experiments-Studies, Monte-Carlo-Simulations or optimizations often generates a heavy computational load. If the simulations themselves are compute intensive, like 3D-FEM or CFD analyses, the execution is normally handed off to a load balancing system, like Sun GridEngine, LSF or PBS Professional. The simulations are then governed by these systems and are executed on dedicated compute farms or high performance computing clusters. When SEE triggers hundreds of analyses simultaneously, sometimes the load triggers system, network or hardware failures, which prevent a simulation from being executed successfully. The ratio of simulations failed due to these non numerical issues can be in excess of 50%, rendering the result of the multi run study unusable.*

To prevent these situations, in this work a component for SEE is developed that interacts closely with the load balancing system. Before the simulation is submitted, the input files are checked for completeness to detect problems in the transfer to the compute servers. If the submission itself fails it is repeated until success and after the return, the log files are parsed for error conditions that the user can specify dynamically. The component furthermore extends the restart ability of SEE such that jobs which are already submitted to the queuing system are not resubmitted again after a restart and expensive redundant calculations are eliminated.

1. Introduction

To ensure successful executions of design exploration algorithms for highly compute intensive tasks using large input and output files in an industrial IT infrastructure, the handling of the computations has to be as robust as possible.

Past experience has shown that among the most common bottlenecks are the following items

- Incomplete transfer of input or output files to the compute server
- Incomplete submission of jobs into the load balancing system
- Overload on the queuing system due to hundreds of individual jobs trying to communicate with the server node simultaneously
- Impossibility to restart an optimization that crashed or had to be stopped
- Impossibility for the user to fine tune job failure criteria, leading to non properly finished jobs polluting the design exploration results.

In this work, the functionality that solves these problems is implemented using SEE the as the overall process automation and orchestration system and PBS Professional as the load balancing system. At SEE side both out of the box components, which are augmented in their functionality using the prologue/epilogue hooks to execute some customized java commands and a specifically developed component are used. At PBS Professional side the Web-Service interface is use to control and monitor the job submission and execution on a very fine level.

2. Connection between SEE and PBS Pro using WebServices

The low level communication between SEE and PBS Professional is implemented using the Web Services provided by the Application Integration Framework (AIF) of PBS Professional..

Web Services are a very abstract way to provide an application programming interface (API) to specific applications. Here the instructions to the application and the necessary data are transported in SOAP-messages typically transported through the HTTP protocol.

On SEE side a new component the "RobustExecutor" is developed that calls these Web Services to submit jobs to the queuing system and to query the system about the job status.

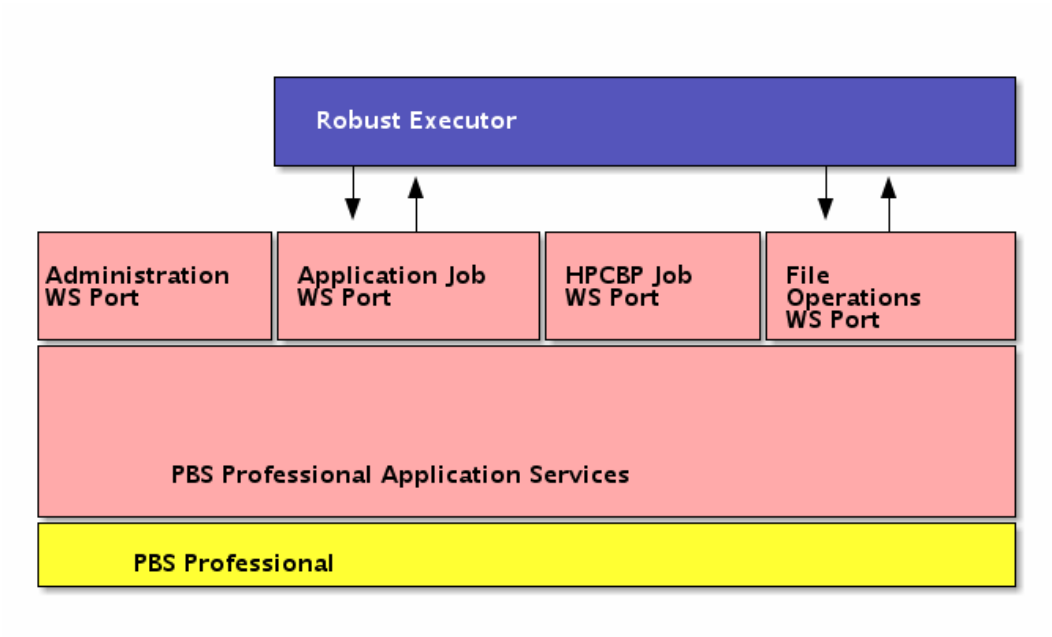

Figure 1 Solution Architecture

The architecture of the solution is shown in figure 1.

This architecture provides the additional flexibility, that the SEE infrastructure and the PBS Professional infrastructure can be implemented on two separate networks, provided that only one HTTP-connection is open between both networks.

3. Development of the Robust Executor Component in SEE

At the side of the Simulia Execution Engine, much of the required functionality, like retrying on execution failure, error handling, logging of analysis output, etc. is already implemented in the OSCommand-Component.

All this functionality can be re-used through the OSCommand-PlugIn provided in the Java-API of SEE.

This out of the box behaviour of eg. the OS-Command-Component or the Simcode-Component is extended through the additions discussed in the remainder of this section.

3.1 File Integrity Check

In many simulation environments the network is constantly under very heavy load. Sometimes this leads to situations, where files are not transmitted properly but are corrupted during the transmission. While this is annoying, but easily fixable in interactive use, by simply re-transmitting the file, when it seems to be damaged, in a fully automated process, this damage check has to be automated.

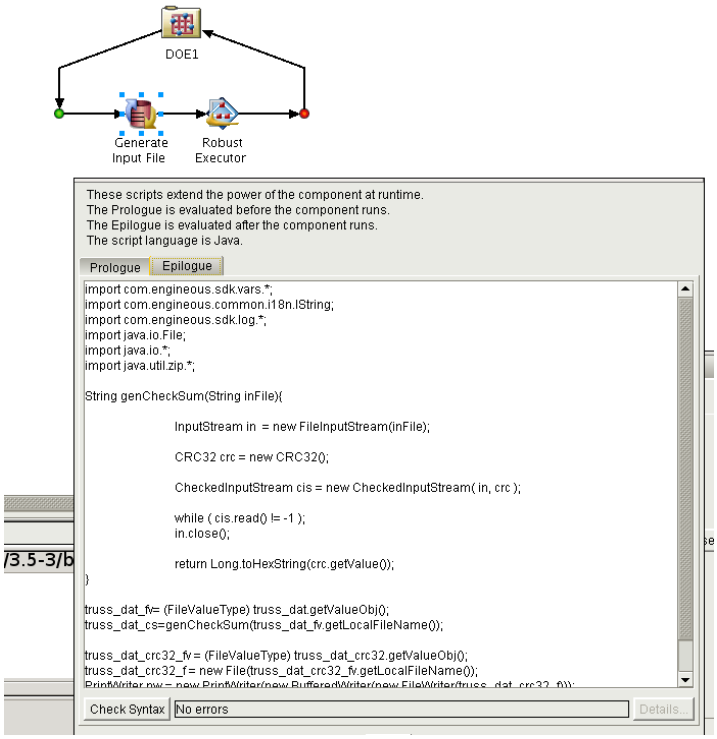

Figure 2 Augmented Data Exchanger Component to Compute the Check Sums

The integrity of the files is checked by computing their CRC32-hash-sum and comparing it against the hash-sum computed at the source, respectively target locations. While in the RobustExecutor component this verification is built in, at the other locations the components that provide or consume the files need to be augmented with some code in the prologue or epilogue that implements this functionality. This is illustrated in figure 2 for a DataExchanger component that generates an input file.

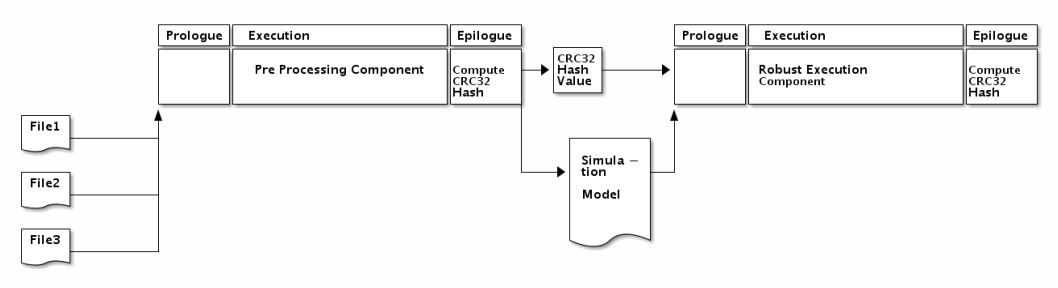

Figure 3 File Integrity Check

The typical usage of this functionality is sketched in figure 3. A preprocessing component takes several input files and assembles them into the simulation model. These files are typically large and have to be copied over to the Robust Execution Component that is executed on the head node of a High Performance Computing (HPC) cluster.

To verify that the file has been properly transferred between the two computers, in the Epilogue of the Pre Processing Component, the hash value of the Simulation Model File is computed and a normal SEE parameter of type string is populated with this value. Both the file and the parameter are mapped into the Robust Execution Component, where the CRC32 hash value of the file, as it has arrived in the local runtime directory, is re-computed and checked against the value of the SEE parameter. If a difference is detected the content of both files differ the execution is stopped and the component is restarted, initializing a new file transfer.

3.2 Check on Proper Submission to the Queueing System

Once all input files have been checked, the analysis can be submitted to the queuing system.

On this step two checks have been implemented. First, only if the return status from the queuing system indicates a successful submission, the execution is continued. Otherwise the submission is re-tried several times.

Next, the component queries PBS Professional for the job just submitted. Only if appears within the job list, the execution sequence is maintained.

3.2.1 Load Reduction for the Queuing System Server

The default behavior of SEE is, that all instantiations of components that run under the governance of a load balancing system directly query the central server of that system, in order to check the status of the jobs. In an industrial environment there might exist situations, where several users run multiple studies at the same time, generating hundreds to thousands of individual components that try to connect to the server. This can leads to an overload at the server.

Therefore the Robust Executor Component delegates this task up to the Design Exploration Technique, under which control it executes. This results in the load on the server being reduced by one or two orders of magnitude, depending on how many jobs are executed at the same time, by the SEE.

As with the file integrity check, the necessary functionality is implemented already in the Robust Executor itself, the Design Exploration component must be augmented by putting a block of code into it's prologue. This code starts a monitoring thread that does the querying of the PBS Professional server. It writes the results, namely the ids of the running jobs, their status and the execution directory, into the local file system. If a job is no longer seen in the queuing system, or has the status finished, the corresponding file will be removed. This is the trigger for the Robust Executor Component to continue its processing.

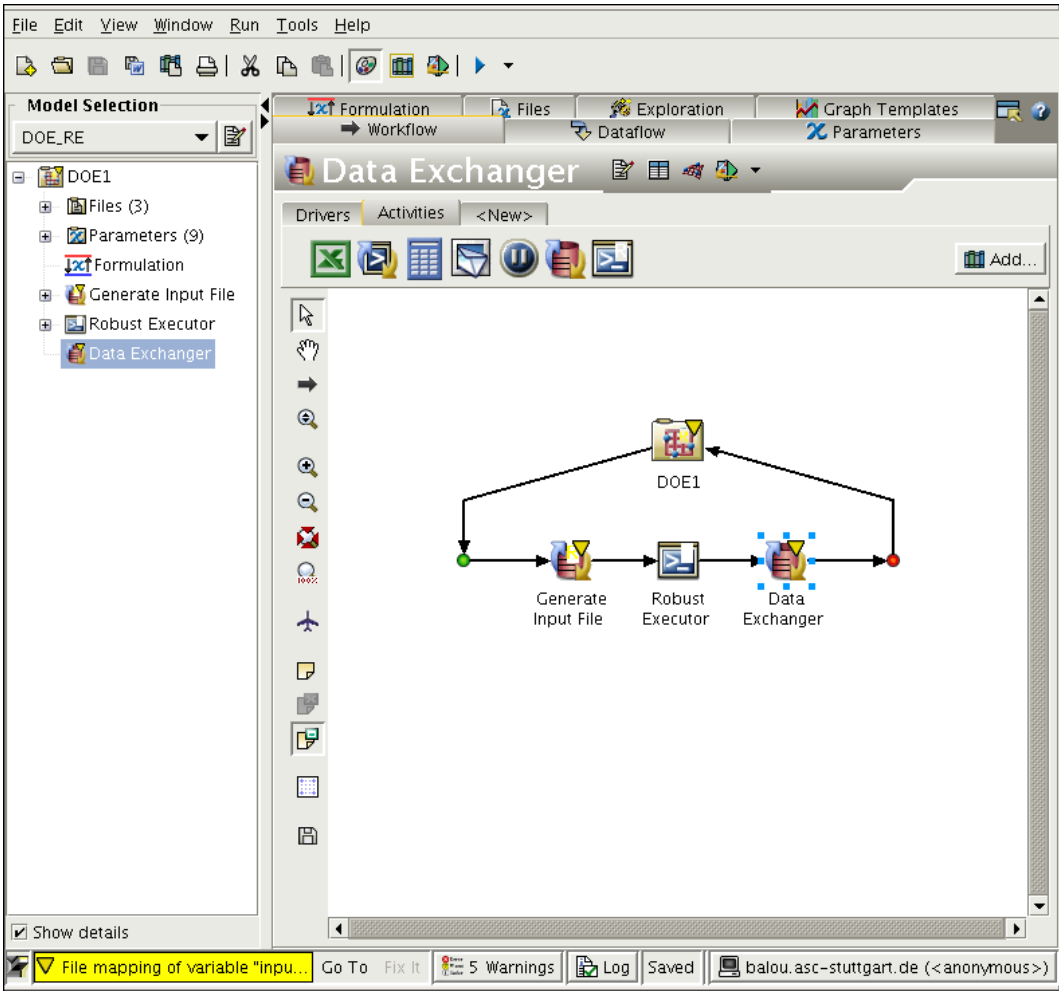

Figure 4 Workflow with the RobustExecutor and Augmented Versions of DataExchanger and DOE components

Figure 4 shows a workflow with an augmented DataExchanger and DOE component to provide the necessary functionality for the RobustExecutor. These augmentations can be done by an administrator once and can be published to the Library, so that the end user does not have to perform this task and concentrate on his own study. In the workflow shown, these components are marked with a white “R” on yellow background.

3.3 Restartability

If the execution of the component has completely finished and all results have arrived in the SEE database, SEE takes care of restoring the results, in the case that the design exploration technique has to be restarted. This is a functionality of the OSCommandPlugIn.

Only for cases when the Robust Executor Component has already submitted the job into the queuing system, but it has not yet returned when the design exploration technique is interrupted, additional functionality has to be implemented.

In order to recover even these orphaned jobs, the CRC32-hash-sums of all input files associated with a specific job are stored into the local file-system together with the job id of the queuing system. Then, after a restart of the design exploration technique, the component checks all it's input files against these hash sums and if it finds that for one job all input files match, it takes the PBS Professional job id from that job, doesn't submit a new job but waits for it to finish. Then the component will copy all output files back from the initial run time directory into its own run time directory and continues with further processing the results.

At the current status of the implementation, this functionality and the load reduction technique assume, that the design exploration technique, and the Robust Executor Component run on the same physical machine, typically the work station of the user that executes the study.

3.4 Dynamic Log-File-Check

One problem with large process automation environments is, that the end user often has very limited control over the quality criteria that decide if the computation was successful or not. If they are too strict, many jobs that could provide good information would be turned down and if they are too loose, jobs that are valid in the general case would pollute the database with unsuited results for the specific study at hand.

The Robust Executor Component let's the users provide a regular expressions in a file that indicate error conditions. These error conditions are tested against all log files that are passed into the component. If one of these regular expressions is matched, the execution of the component is terminated and the results are marked as useless.

4. Conclusion

While SEE provides already out of the box a rich and flexible environment for controlling the execution of long running simulation analyses, for some specific requirements this functionality needs to be extended.

Within a project at the Automotive Simulation Cluster Stuttgart, such an extension has been programmed, that allows SEE to execute compute intensive automotive crash simulations to be executed very robustly on a compute farm. Also the load that very many instantaneously running simulations pose on the server of a queuing system has been reduced and a mechanism allowing for recapturing orphaned runs from a previous execution of an interrupted design exploration technique has been implemented.

These additions need a much closer communication between the SEE-component and PBS Professional, than what is available per default. This has been implemented using the AIF Web Service interface of PBS Professional.

Owing to the modular architecture of SEE all originally available functionality in the standard SEE components can be reused in the new developed component. Already existing components can be augmented easily in their functionality to work together with the new component, by placing small blocks of Java-Code into the prologues and epilogues of these components.

5. Acknowledgements

The authors would like to acknowledge the support from the Automotive Simulation Center Stuttgart.

The authors would like to thank Ralf Rehburg and Jochen Seybold of Altair Engineering GmbH for their support with PBS Professional and Stefan Schwarz of Dr. Ing. h.c. F. Porsche AG and Franz R. Klimmetzek of Daimler AG for their valuable input and requirements specification.