ANALYZING BIG DATA WORKLOADS USING DISCRETE SIMULATION

Stephan Kiemle, Julian Meyer-Arnek, Nicolas Weiland

German Aerospace Center DLR

ABSTRACT

Virtualized IT platforms abstracting from physical infrastructure offer easy scaling of storage and compute resources. This is a great progress for handling dynamic big data workloads but holds the danger of solving performance issues just with upscaling, without looking at causes. We propose to analyze the behavior of complex systems handling big data workloads using system modelling and discrete event simulation. It can be very revealing to see a simulated system in action by running simulations in different configurations, showing the impact of a modified system structure on its runtime behavior and resource consumption. With a tool-based system workload simulation example in the Earth Observation data domain we show how this approach can lead to significant resource and cost savings.

Index Terms— system modelling, grey-boxing, discrete event simulation, infrastructure optimization, scalable platform, performance prediction

1. INTRODUCTION

In the domain of Earth Observation as in many other high-performance data analytics applications, huge amounts of data need to be stored, accessed and processed. The workload placed on networks, compute nodes and storage systems is thereby often difficult to predict. The dynamic of user requests, the different size of data units and the complexity of processing algorithms makes a static resource allocation very difficult. Virtualized IT platforms offered in on-premise computing centers as well as by public cloud providers allow to quickly react on bottlenecks by adding storage, compute and network resources almost on-the-fly. But are these (often expensive) resources really necessary in the sense that they perfectly fit to a specific need within a complex system configuration in order to reach overall system throughput, timeliness or cost target requirements? Wouldn't another workflow configuration and set of resources do the same job even faster and with less cost, meaning better energy efficiency and smaller carbon footprint?

In this paper we demonstrate how a system model can be used to run repeated workload simulations in order to identify system bottlenecks and to optimize IT resource assignments as well as system workflows. We show that simulation produces fast and reliable results, without need to spend efforts in real large-scale performance experiments.

We will first introduce a specific approach for effective modelling of complex systems in their static structure as well as their dynamic data and control flow. Based on the system model we will show how to conduct and analyze iterative simulation runs in so-called "what-if-experiments". Finally, we will demonstrate the optimization of a complex system used for the large-scale management and processing of a long Earth Observation data time series, step-wise enhancing the initial system configuration already before deployment.

2. SYSTEM MODELLING

The typical steps of achieving a reliable system model follow the bottom-up reverse analysis approach. We start at the physical layout identifying elements of infrastructure, IT hardware, network, system and application software. In the next step a logical system layout can be derived, grouping functional elements into larger units in multiple levels and connecting these typically along "uses/used-by" associations (Fig. 1). The logical composition formalizes the logical layout by introducing model abstraction levels and cardinalities of model elements for which similar properties and a similar behavior can be assumed.

This reverse analysis approach does not rely on top-down system specification, such as its documented architecture and design, as this theoretic information usually does not reflect a system in its final implementation state with sufficient detail. Since a physical layout may not always be applicable, e.g. for system and application software, a logical layout can also be directly defined based on software being deployed in different unit levels, e.g. libraries/modules, packages, components, containers.

2.1. Static Model Elements

All elements in the logical composition, at any model level, are valid model elements. The leaf elements typically have specific properties describing their main influence on the overall system behavior:

- A compute element consumes a load-dependent electrical power and runs with a certain clock frequency.
- A memory or storage element has a capacity, an I/O bandwidth, an access latency and a power consumption.
- A network element has a nominal throughput and a latency.
- All these elements have a typical breakdown probability and mean-time-to-repair distribution.

The model elements in higher levels aggregate the behavior of their sub-elements which can be expressed with similar properties. They usually also have additional properties e.g. reflecting their state and capacity constraints at their respective level. Properties with variable values have to be configurated with a probability distribution to reflect their variance in the real system.

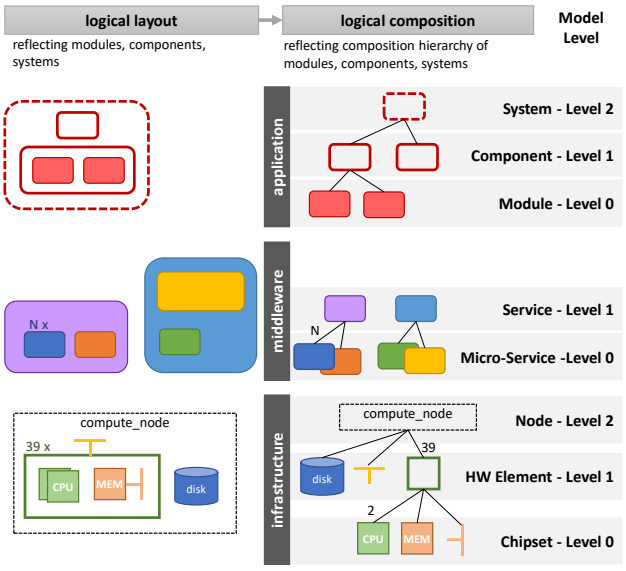

Fig. 1. System modelling

2.2. Grey-Boxing

Building a model reflecting the system in all its details is totally unrealistic. Most IT-systems have a complexity which does not allow capturing all elements with their exact properties as required by a "white box" analysis. On the other hand, fixing a system model at a certain higher level considering all elements of this level as "black boxes" will usually result in either a rather trivial high-level model or a model with some elements having a trivial deterministic behavior and other elements with a not explainable, non-deterministic behavior hiding the relevant structures with major impact on runtime and resource consumption.

The grey-boxing approach, also well-known as targeted software testing concept [1], combines "white box" and "black box" modelling by opening sub-elements only to that extent which is necessary to represent the system in all those parts which are expected to have significant impact. The result is a model with a heterogenous level of detail, using elements with low impact at high level and detailing elements to lower level as necessary to explicitly reflect the behavior of really relevant parts.

The grey-boxing approach requires not only a detailed knowledge of the system in all its levels but also experience on which parts typically have a relevant impact [2]. This may also depend on the workflows and scenarios which shall be simulated using this model later, as different workflows may require system functions at different extent, so that different parts of the system would need to be further detailed in the model [3]. A corresponding iterative refinement of the model based on first simulation results can help to reach a "grey box" model with sufficient level of detail for all scenarios it is intended to support. A model coarsening should also be performed if the first simulation results show that parts of the model have no impact at all. The "grey box" models of the same system may be different when targeted to different system usage scenarios. The lesser scenarios are to be covered, the simpler the adequate model will be.

In difference to the leaf model elements introduced in the previous section, higher level "grey boxes" are typically characterized with the following behavioral properties:

- Number and size of input data units processed (or stored or transferred) per time interval
- Number and size of output data units produced per time interval
- Compute, memory and power resources consumed
- Breakdown probability and mean-time-to-repair distribution

2.3. Dynamic Model

A simulation is supposed to execute a dynamic activity on a pre-defined static system model. As with the static system model discussed so far, the dynamic activity needs to be modeled by analyzing e.g. the steps of workflows, applying the "grey box" approach, down to the level required to represent partial activities (steps) with major impact on latencies, runtimes and resource consumption.

The dynamic model elements can typically split into two groups, those with a high impact and those with a low impact but required to connect the dynamic model elements with "triggers/waits-for" associations in order to be able to run complete end-to-end workflow simulations.

Dynamic model elements with high impact are e.g.

- Upload/download/store data, impact by the volume of data units transmitted
- Process data, impact by the complexity of the processing and the volume of data processed

Dynamic model elements with low impact but needed for connecting dynamic elements are e.g.

- Send/receive message
- Allocate storage/compute resource
- Deploy/launch software
- Release/remove data
- Release storage/compute resource

The static and dynamic model elements need to be aligned in their level of detail so that the dynamic elements fit onto the system elements which are supposed to handle them. For example, a processor allowing the parallel processing of n parts of an input data product can be modeled with sub-elements deployable on n processing nodes each handling one of n sub-elements of one data product.

3. SIMULATION SETUP

The model definition and simulation setup shall be demonstrated with an example system based on DLR experience in EO payload data ground segments [4]. It consists of a Long-term Archive (LTA) hosting a very large data record of level 1B (L1B) atmospheric spectrometer data which shall be reprocessed to multiple level-2 (L2) atmospheric composition data products. The LTA is connected via WAN to a cloud platform, on which this reprocessing shall be performed.

The provision of the L1B data onto the platform is controlled by an operator through the submission of bulk reload requests covering e.g. one sensing year. The Ingestion system processes these bulk requests by downloading products from the LTA using parallel transfers, verifying data consistency, extracting metadata and placing the L1B products on an online storage. The Processing system polls the L1B input products and automatically launches the L2 processing in subsequent batches of e.g. 2 weeks sensing time length.

Each of these batches needs to be executed in three consecutive passes. Pass 1 processes each input product to intermediate output products required as input to the following passes. Pass 2 processes only one product per input sensing day in order to perform some background correction. Pass 3 processes all input products again using the intermediate processing results with background correction information generating all expected final output products.

It is important to note that the three passes need to be processed sequentially but that within each pass, all products can be processed in any order and in parallel.

The demonstration system has been modelled using the discrete event simulation tool JaamSim [5]. This very powerful open source tool was designed for modelling and simulating real-world production lines, but it proves to be also very well suited for modelling IT infrastructures and software applications, and simulating digital control and data flows.

In the JaamSim tool, the demonstration system is modelled using entities, e.g. *L1B-Product*, *Pass1Request* (Fig. 2), circulating through a process flow (dynamic model) visualized with blue arrows, composed of different processing, queuing, conveying and branching stations (static model). Stations are configured with properties such as the time required to process an entity. Various probability distributions can be used to model uncertainty e.g. for processing time, the occurrence of unplanned outages and the time needed to recover. The dynamic process flow is customized using thresholds allowing to model water marks and capacity limits.

In this paper, we can focus only on a small part of the implemented demonstration model. This part consists of the data-driven lookup of data batches on an *OnlineStorage* resource. As soon as a full input data batch is available, a *L1B-Batch* entity is created and passed to the *BatchWaitQueue* (left in Fig. 2). It is picked from there by a server *BatchProcControl* which is limited by a resource ensuring that only one batch request is processed at a time. The server creates processing requests for the three different passes 1/2/3, parameterized with individual processing time. These get only released onto the central L2ProcessingQueue, if the previous pass was totally completed.

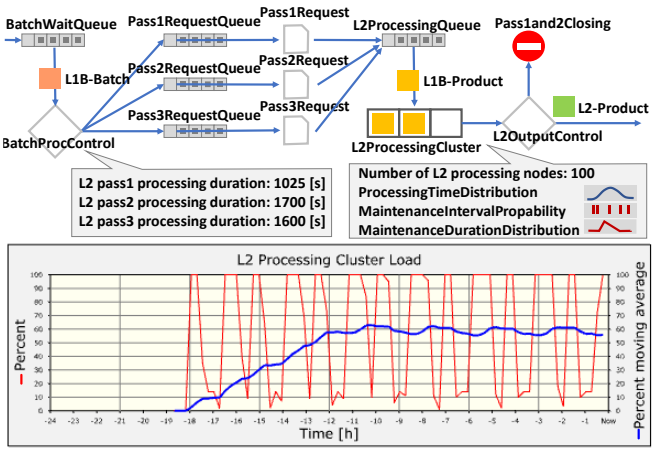

Fig. 2. System simulation with JaamSim (extract)

The *L2ProcessingCluster* processes all pass 1/2/3 requests waiting in the *L2ProcessingQueue*. This cluster has a configurable number of nodes for parallel processing, as well as other modifiable parameters modelling uncertainties regarding the processing duration variance and the cluster maintenance downtime. The *L2OutputControl* server detects the final pass 3 output *L2-Products* and passes them on the *OnlineStorageL2*, and triggers the closing of the batch processing request.

4. DEMONSTRATION

4.1. Simulation Parameterizing

After the initial definition of a "grey box" static and dynamic system model, a specific workflow can be mapped onto the model using the following steps:

- Define a start condition, e.g. availability of initial input data, availability of an initial processing request
- Parameterize all model elements, especially the dynamic elements, e.g. data sizes, processing durations, probabilities, downtimes for maintenance activities and un-planned outages, as partially shown in Fig. 2 for the *L2ProcessingCluster*
- Complete the connection of model elements as required by the workflow

In the demonstration example, multiple input parameters have been defined to easily change the settings for simulation runs executed using this model. A part of these parameters is visible above the model in Fig. 2. During simulation runs being executed, the JaamSim tool collects very detailed statistical information on each dynamic entity instance

flowing through the system as well as on each process flow station handling entities. During and after simulation, each object can be inspected to analyze e.g. its load and the average queueing or processing time. Moreover, any of these measurements can be logged for later analysis and visualized directly in text fields and diagrams within the model as shown with the load curve in Fig. 2.

4.2. Analyzing Simulation Runs

Multiple simulation runs with different input parameter values, so called "what-if-experiments", can be performed to observe the system while executing and measuring the behavior of any part within the model as well as the overall simulation duration. The speed of a simulation run can be accelerated by multiple dimensions so that results become visible and analyzable after reasonable time even if the simulated scenario would require weeks or months in real-time.

The demonstration system simulation run shown in Fig. 2 analyzes the load of the *L2ProcessingCluster*, aiming at maximizing it for best resource usage. For a series of simulation experiments listed in Table 1, we assume that the processing cluster can provide a maximum of 120 nodes and that a fix cost rate per node is applied.

Table 1. Experiments and cluster load results

Experi-ment	Input Products per Batch [n]	Processing Nodes [n]	Resulting Average Cluster Load [%]
#1	210	100	60% (see Fig. 2)
#2	210	105	67%
#3	1050	105	90%
#4	1050	70	95%

Working with a batch sensing time period of 14 days, corresponding to 210 input products, and a capacity of 100 processing nodes for parallel processing in the cluster shows, after some ramp-up time, an average cluster load of 60%, which is far from being optimal (experiment #1 in Table 1). Increasing the number of processing nodes only by 5 already significantly increases the cluster load to 67% (experiment #2). This can be explained by the fact that the number of input products processed in pass 1 and pass 3 is a multiple of the number of processing nodes (210 = 2 x 105).

A third "what-if-experiment" (experiment #3) using a batch sensing time period of 70 days increases the average cluster load to 90%, because still the number of input products is a multiple of the number of processing nodes, and pass 2 uses now 70 of the 105 nodes, which is better than using only 14 of 105 nodes in the previous experiment.

We are still able to increase the cluster load to 95% when running experiment #4 with 70 instead of 105 nodes, because then also pass 2 almost uses the full cluster resources. We will not be able to achieve a higher cluster load, because the model uses a variable individual processing duration using a normal probability distribution so that not all processing runs terminate at the same time. Moreover, the model defines regular cluster maintenance downtimes with variable duration following a triangular probability distribution, again slightly reducing the usage rate of the cluster being understood as a continuous service.

This is a good example of using simulation to find a system configuration which suites best to the specific target within the limits of possible system configurations. In this case we could significantly reduce the overall system cost by use of 70 nodes instead of 120 nodes, without major impact on the total reprocessing duration.

5. CONCLUSION

The approach of predicting workflow duration and resource consumption using simulations has the main advantage that statements can be made before any expensive setup and test of systems or cloud resources, and that we can experiment with multiple system setups in order to find a setup which best fit to the planned specific workflow or expected mix of system usage scenarios. The main disadvantage of this approach is that the precision of predictions strongly depends on the quality of the system model. Models should portray the system in heterogenous levels detailing the system components which have most expected impact on performance (grey-boxing). They also should be parameterized with probabilistic distributions to reflect randomized behavior as observed in real complex systems.

Model definition and parameterizing requires deep knowledge and experience on the system components and their typical behavior. Based on such models, simulations can be executed multiple times and using different parameterizations, and the system model can easily be tuned proposing adaptations within the limits of acceptable system changes, in order to optimize the overall system throughput and cost/energy efficiency.

REFERENCES

[1] Z. J. Li, H. F. Tan, H. H. Liu, J. Zhu and N. M. Mitsumori, "Business-process-driven gray-box SOA testing," in IBM Systems Journal, vol. 47, no. 3, pp. 457-472, doi: 10.1147/sj.473.0457, 2008

[2] H. Al-Sayeh, S. Hagedorn, K.-U. Sattler, "A gray-box modeling methodology for runtime prediction of Apache Spark jobs", Distributed and parallel databases, ISSN 1573-7578, Vol. 38, 4, P. 819-839, https://doi.org/10.1007/s10619-020-07286-y, 2020

[3] H. Al-Sayeh, M. A. Jibril, B. Memishi, K.-U. Sattler, "Blink: lightweight sample runs for cost optimization of big data applications", New Trends in Database and Information Systems, P. 144-154, https://doi.org/10.1007/978-3-031-15743-1_14, 2022

[4] S. Kiemle, K. Molch, S. Schropp, N. Weiland, E. Mikusch, "Big Data Management in Earth Observation", IEEE Geoscience and Remote Sensing Magazine (GRSM), 4 (3), pp. 51-58. Geoscience and Remote Sensing Society, ISSN 2168-6831, doi: 10.1109/MGRS.2016.2541306, 2016

[5] JaamSim Discrete Event Simulation Software, https://www.jaamsim.com