Cost-Effective Sizing of Your HPC Cluster for CAE Simulations

Nils Henkel[1], Sebastian Treiber[1]

[1]GNS Systems GmbH

1 Abstract

Computer aided engineering (CAE) is a very important part of product development, especially in the automotive industry. At the core of each CAE department stands a compute cluster which is used to run mostly finite element solvers in order to simulate car crashes, airbag inflation and many more physical processes. Almost every software vendor for CAE solvers uses a different license model. Especially the formula for the token scaling (number of tokens as a function of the CPU cores used) is unique for each solver type.
In this work, we present a method to derive the total costs of a single solver run and propose possibilities to increase the efficiency of small to medium-sized CAE clusters. Instead of deriving general instructions, we show how specialized analysis of license models, solver behavior and hardware options can be used to maximize efficiency.

2 Introduction

The use of CAE techniques as an integral part of product development has become indispensable in the automotive industry. The investigation of different problems with the help of virtual simulation is used daily by development teams. They need considerable computing power, which is provided by HPC (High Performance Computing) clusters. Typical cluster installations in the automotive industry can range from local resources with 50 to 200 cores up to centralized cloud computing installations with more than 10,000 cores. However, in general, the requirements for the HPC systems are higher than available computing power. This has economic (limitation of licenses) and infrastructural (limited cooling facilities and space in the data center) reasons. Therefore, an optimized schedule of the calculations in batch operation is essential in order to achieve maximum efficiency of calculations.

GNS Systems is an IT services company specializing in these kinds of challenges. In this contribution we would like to give examples about how we achieve more efficiency by detailed and scientific analysis of the situation given by the customer.

First, using benchmark results on modern Intel Xeon E5 Processors, we investigate how the performance of various CAE software depends on the number of CPU cores used and on the details of the CPU core binding. The positive effect of the turbo frequency can be clearly observed. Second, we derive a general formula for the cost of a simulation (including hardware, software, and personnel cost). We show that in situations that are typical for normal commercial users in the automotive industry, there is often a "sweet spot" of CPU cores. We define a sweet spot as a number of CPU cores other than 1 which minimizes the total cost of a simulation. Lastly, we discuss possible tricks (such as underloading of hardware) which a typical industry user could make use of in order to reduce the total cost of simulations while leaving the total capacity of his cluster unchanged.

3 Theoretical considerations and derivation of a formula

In this section we derive a universal formula to calculate the total cost of a CAE job. There are three main factors which determine the total costs. These are:

1. License costs
2. Hardware costs
3. Additional factors like personnel costs, general costs for the waiting time until a solver job has finished or the cost incurred by having to delay a project

3.1 License costs

Each software has its own license model. In this work we will use the solvers LS-Dyna, Abaqus, and Ansys as examples.
Usually a cluster jobs needs a number of tokens which is calculated from the number of used cores only. In addition, some software vendors offer different kinds of tokens that can be combined to allow multi-core calculations. Ansys, for example, offers “anshpc” and “pack” licenses.
Table 1: gives an overview over the token scaling and costs of the respective software.

Software	Tokens T per number of cores n	Annual costs in €/token	Costs per second
LS-Dyna	T(n) = n	1000	31,7µ€
Abaqus	$T(n)=5\cdot n^{0.422}$	2500	79,3µ€
Ansys	$T_{meba}(n)=1; T_{hpc}(n)=(n-2)$ or $T_{meba}(n)=1; T_{hpc}(n)=([\ln(n/8)/\ln(4)]+1)$	meba: 14000 hpc: 1700 pack: 9000	476µ€ 38,0µ€

Table 1: Overview of token scaling and annual costs for different CAE solvers.

Fig.1: shows the number of tokens needed as a function of the number of cores used for each considered solver. For Abaqus und Dyna, the relative token cost is identical to the number of tokens. For Ansys, where different tokens types are available, we take the relative costs into account in order to give a reasonable scaling behavior.

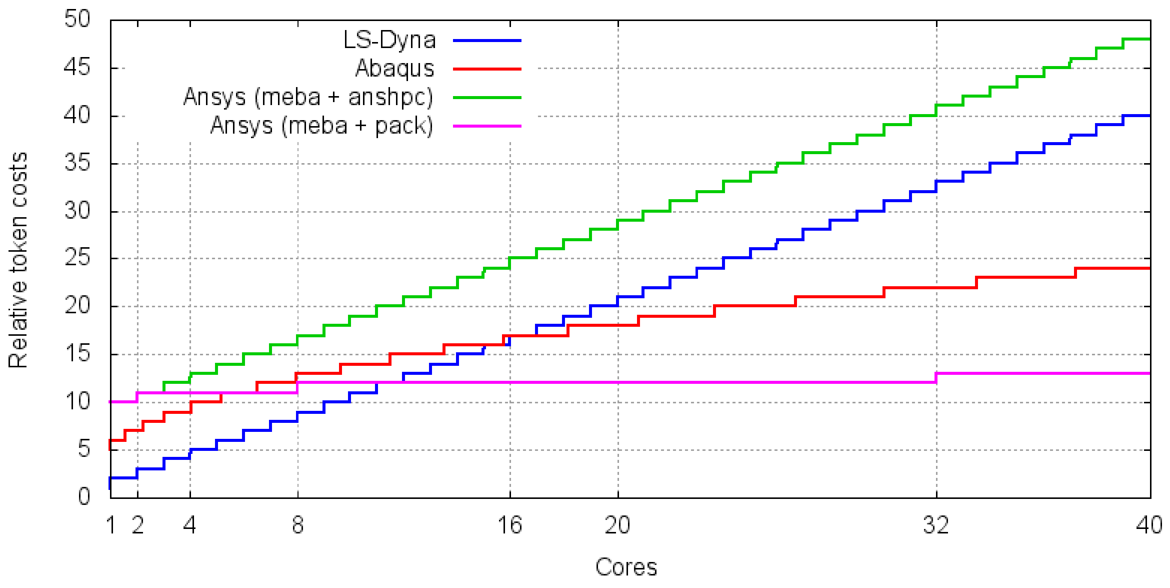

Fig.1: Number of tokens needed as a function of the CPU cores used for three different solvers. The absolute costs of the different tokens do not need to be taken into account for Dyna and Abaqus. Only for Ansys where different token types are available we take relative costs into account.

3.2 Hardware costs

CAE jobs usually run on enterprise rack servers. There is a large variety of hardware and in particular CPUs. In this work we chose a typical server setup to estimate the costs. Note that these costs may vary by at least a factor of two.
Table 2: gives an overview of typical hardware costs for a one machine of a small to medium-sized CAE cluster.

Purchase cost for one 16 core machine	9000 €
Depreciation schedule	5 years
Costs per year	1800 €
Costs per second and core	3,57 µ€

Table 2: Overview of hardware costs for a single machine in a small to medium-sized CAE cluster.

3.3 Additional factors

In addition to the costs described above, there are additional costs which are much more difficult to estimate, such as personnel costs. In the situation where an engineer submits a job and carries on with other work, there is almost no extra cost. Assuming, however, that the engineer waits for critical results and cannot do useful work while the job runs, we would have to add a large cost factor.
Another type of cost that is hard to estimate in general but should not be neglected when deciding for a cluster are operating costs (mostly for electricity and space in a datacenter).
Thus, we will not consider these kinds of costs in a quantitative sense, but we do emphasize that it is important to take these costs into account in order to get a meaningful total cost analysis.

3.4 Deriving a universal cost formula

In this section we will use the following abbreviations:

n: Number of cores which a given job uses
c: Total cost of the job in €
c_l: License costs in €
c_h: Hardware costs in €
l(n): License costs for n cores in €/s
h: Hardware costs in €/(s · cores)
t: Runtime of the job in seconds

The total costs of a job can be written as:

$$\begin{aligned} c(n) &= c_h(n) + c_l(n) \\ &= t(n)nh + t(n)l(n) \\ &= t(n)\left(nh + l(n)\right) \end{aligned}$$

The scaling behavior of the quantity l(n) is described in section 3.1 and plotted in Fig.1:.
The quantity t(n) describes the scaling behavior of the solver itself. Since parallel computing is a very complex field it is impossible to give an exact formula here. Therefore experimental results are used to obtain the desired information.

4 Experimental benchmark results

In this section we show experimental results regarding the scaling behavior of typical CAE jobs. All tests have been performed using comparatively new enterprise rack servers. However, since access to CAE machines is limited we could not use the same hardware for all tests. Nevertheless the results can be used to extract qualitative results.
In order to get a meaningful quantity for comparison we introduce "Token Hours" (TH). These are given by the total amount of time in which a number of tokens are occupied by the job being considered. TH is defined as follows:

TH = Number of used tokens (T) × Runtime of the job in hours (H)

Both, T and H depend on the number of CPU cores used. T(H) can be calculated using the formulas from Table 1:. H(n) is acquired from experimental results.

For the benchmarks in this section we used one (arbitrary) test job for each solver. Obviously the scaling behavior differs for different types of jobs. For example, we expect large differences between explicit and implicit jobs. We point out that the results we show here are not universally valid, but act as examples on how the costs for a solver run can be evaluated. At the end of this work, we show how to optimize performance using these examples. The goal is to demonstrate methods and not to give detailed universal instructions.

4.1 LS-Dyna

The benchmarks shown in this section have been performed on an Intel Xeon E5-2690 v2 CPU.
Fig. 2 shows TH(n) for a simple LS-Dyna test job (neon refined revised from [TopCrunch]). The result is a roughly linear increase of TH with increasing CPU number (n). Only between 2 and 4 cores the token hours are identical.

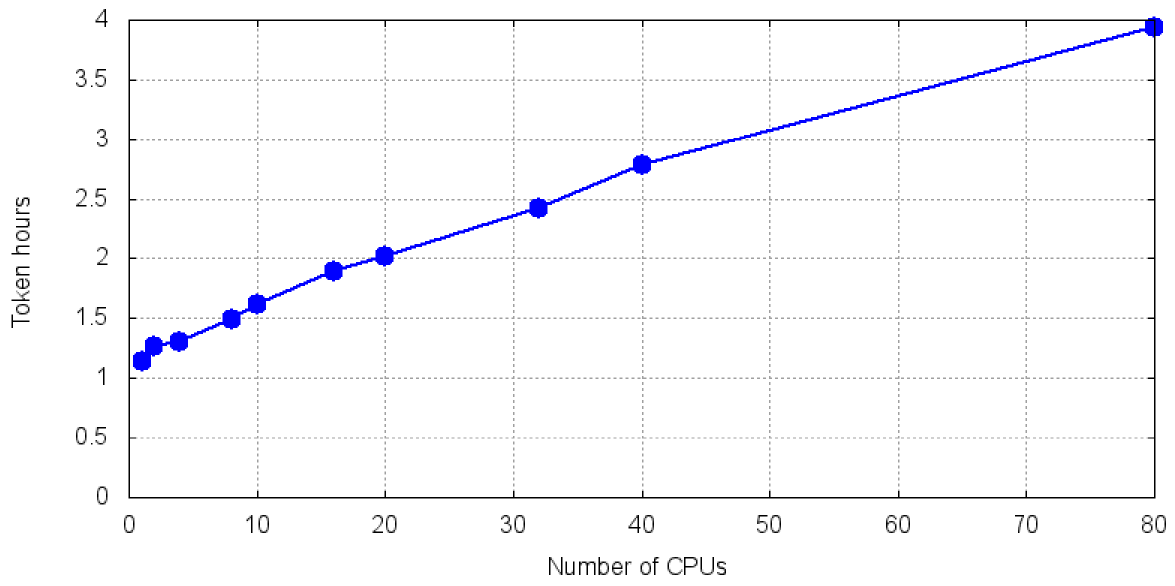

Fig.2: Token hours (TH) versus number of CPU cores (n) for the LS-Dyna test job. TH(n) increases almost linearly for increasing n.

4.2 Abaqus

The benchmarks shown in this section have been performed on an Intel Xeon E5-2690 v2 CPU.
Fig.3: shows TH(n) for a simple Abaqus explicit test job (e1).

TH drops very fast between one and four cores. For more than four cores TH decreases slowly but monotonically.

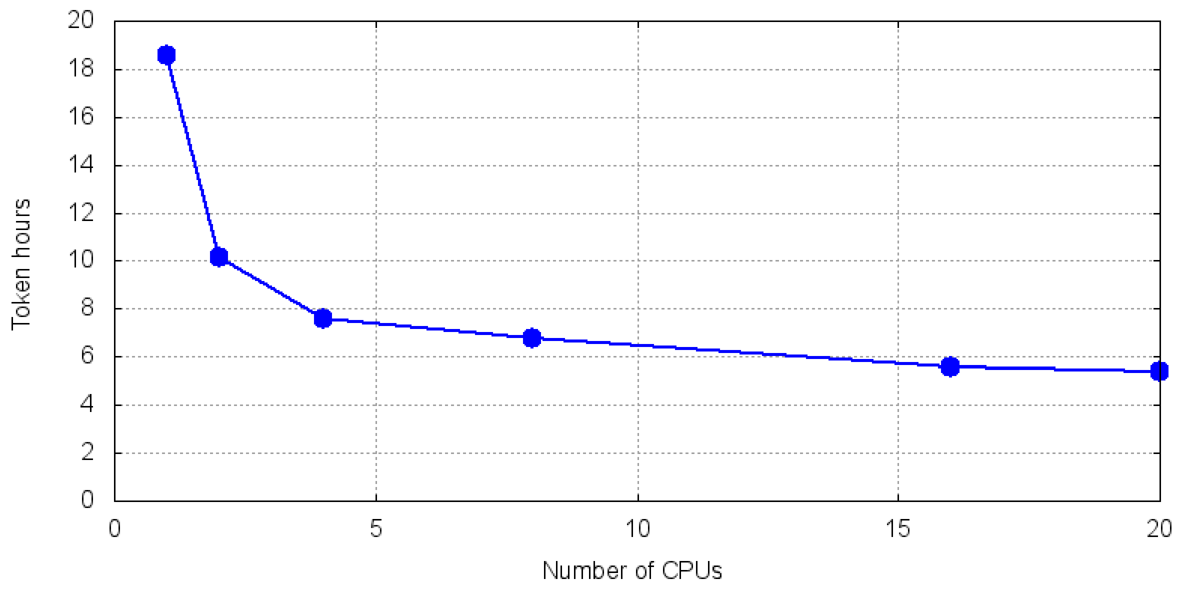

Fig.3: Token hours (TH) versus number of CPU cores (n) for the Abaqus test job. For increasing CPU numbers, TH decreases significantly.

4.3 Ansys

The benchmarks shown in this section have been performed on an Intel Xeon E5-2667 v2 CPU. Fig.4: shows TH(n) for the Ansys test job.
For up to four CPU cores, TH decreases very fast. There is a non-distinct minimum at around 8 CPU cores. The curve is roughly flat between 4 und 16 cores.

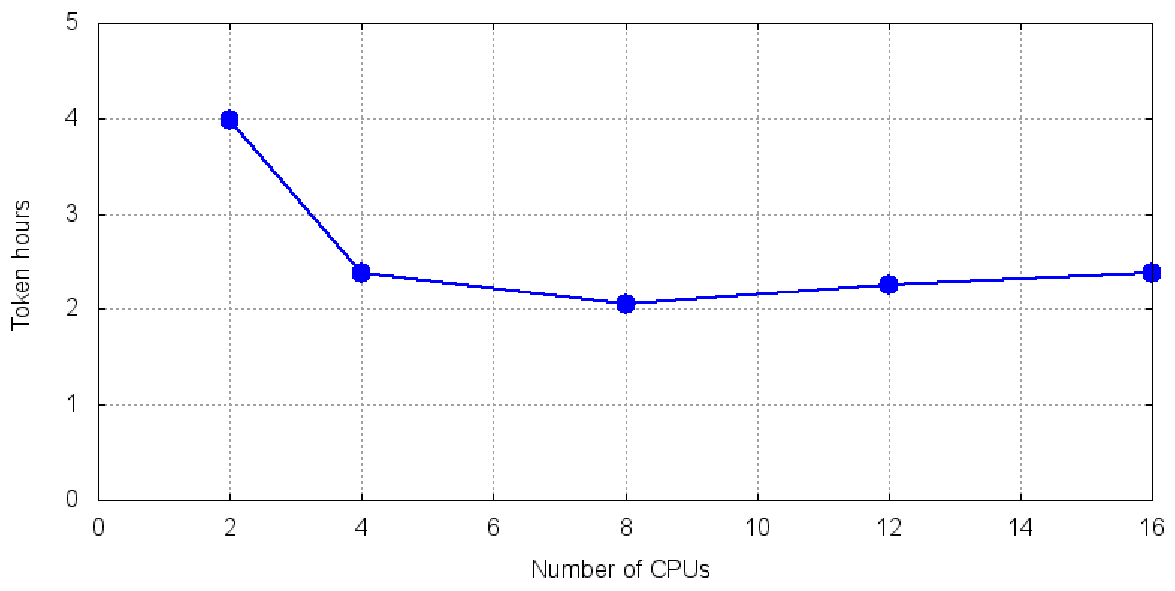

Fig.4: Token hours (TH) versus number of CPU cores (n) for the Ansys test job. There is a non-distinct minimum around 8 CPU cores.

5 Total cost analysis

In this section we add up all the previously shown data and theoretical considerations. That means we will calculate the total costs for the given test jobs using equation (1). This calculation combines hardware and license costs (see Table 1: and Table 2:) with the CPU core scaling of the respective solver (Section 4).
Fig. 5 shows the results for the LS-Dyna test job, Fig. 6 shows the results for the Abaqus test job and finally, Fig. 7 shows the results for the Ansys test job.

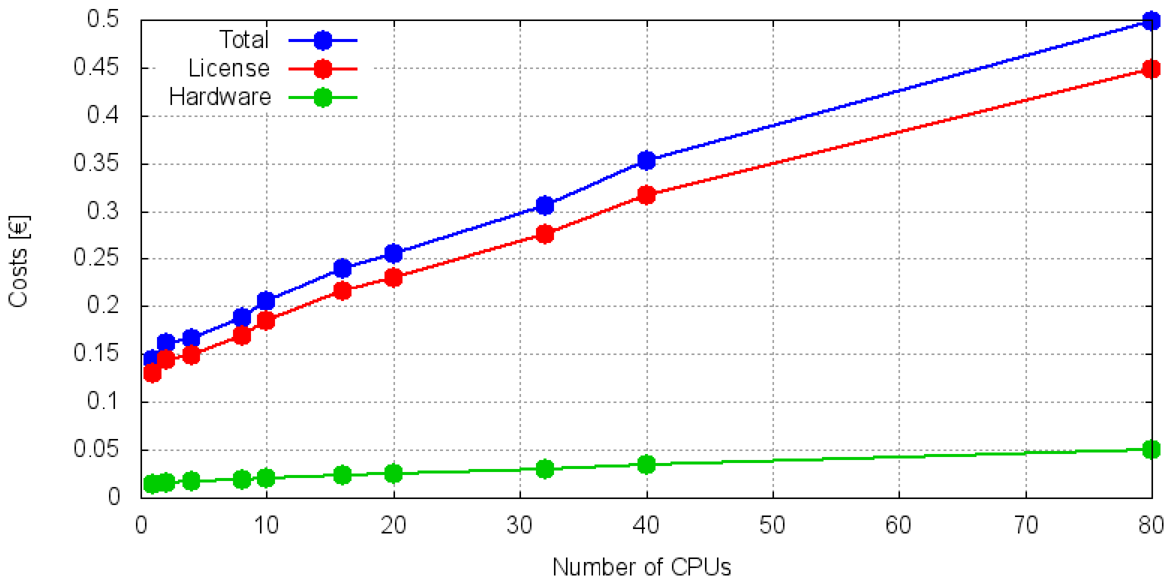

Fig.5: Costs for the LS-Dyna test job as a function of CPU cores used.

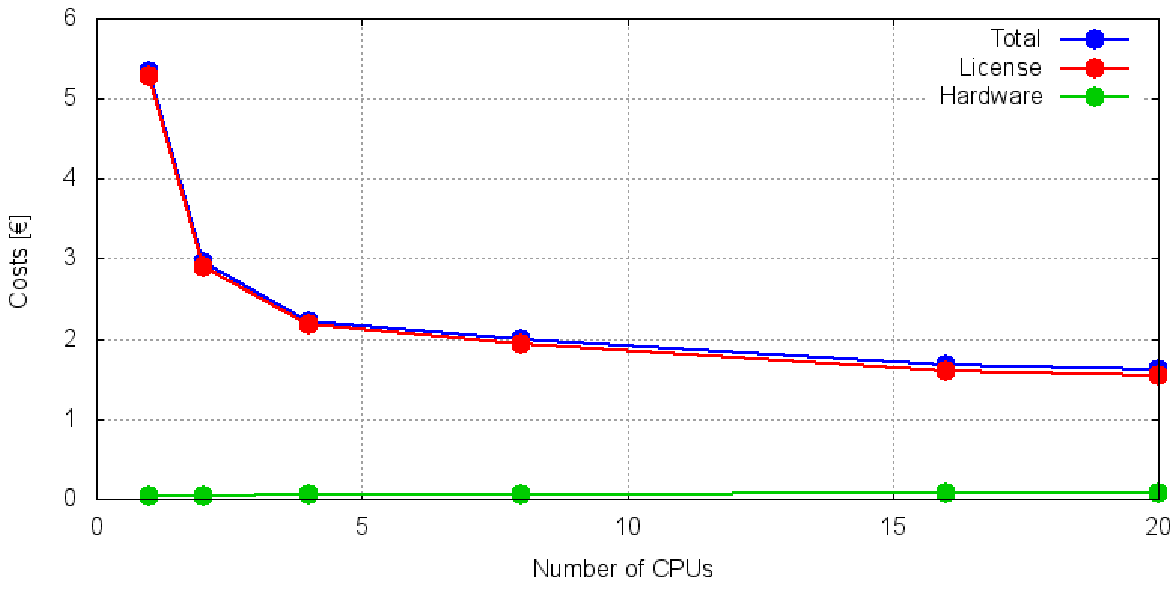

Fig.6: Costs for the Abaqus test job as a function of CPU cores used.

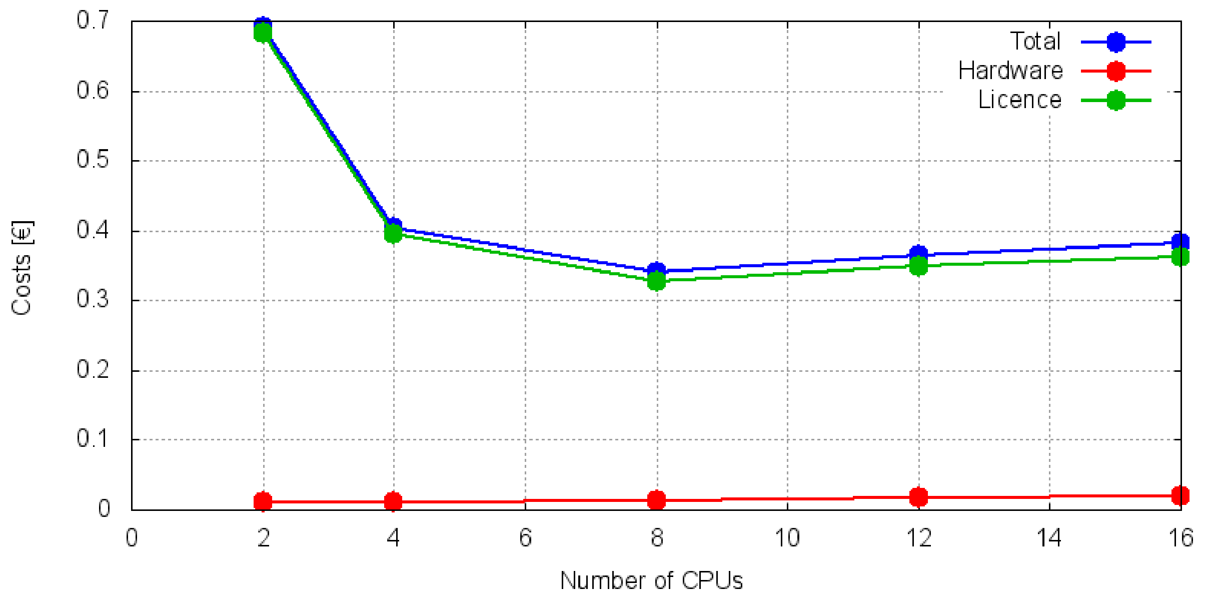

Fig.7: Costs for the Ansys test job as a function of CPU cores used.

6 Methods to maximize performance

In this section we show how to use the given data to maximize the efficiency of a given job. We start with general considerations in section 6.1. In section 6.2 we present a special method, namely hardware underloading to increase cost efficiency.

6.1 General considerations

Analyzing the figures from section 5 leads to two important results.
First, all figures in this section have one distinct feature in common: The hardware costs are significantly lower than the license costs which is a crucial insight for anyone who wishes to improve the cost efficiency of their simulations. There is roughly a factor of ten in costs between these two items. That means the total cost of a CAE batch job is determined mainly by the license costs. Accordingly, it is crucial to avoid hardware shortage which would lead to idle licenses. On the other hand, having some idle hardware is often a very small price to pay if it leads to more efficient license usage.

Second, although hardware costs are negligible, it is possible to increase the overall performance of a CAE cluster by purchasing a CPU which fits best to the solver being used since the right CPU will maximize license efficiency.
LS-Dyna: Fig.5: shows the total costs as a function of the number of CPU cores used for the LS-Dyna test job. Increasing the number of CPU cores always leads to an increase of the total costs. This means on the one hand, that the use of more than one CPU is only reasonable if additional factors (such as the costs for the waiting time until the job is completed) play a role. On the other hand it is crucial to have a CPU with a high clock rate per core. Since the use of each additional core increases costs significantly, one wants to have as much power in one core as possible.
Abaqus: Fig.6: shows the total costs as a function of the number of CPU cores used for the Abaqus test job. Increasing the number of CPU cores always reduces the total costs of the test job. That means it is desirable to use as many CPU cores as possible. In addition, one wants to include as much computing power as possible in one server in order to keep the total costs low. I.e. hard-drives, chassis, and so on are shared by all CPU cores in one server. Therefore, the best choice

would be a CPU with 10 or more cores with comparatively low clock rate but high overall performance.
Ansys: Fig.7: shows the total costs as a function of the number of CPU cores used for the Ansys test job. The total costs drop quickly between two and four cores, they show a non-distinct minimum at around eight cores and increase slowly for higher core numbers. Due to the more complex license model, the conclusion contains different aspects. First of all, each Ansys job needs one relatively expensive general token (for example "meba"). In order to use multiple CPU cores, Ansys provides different possibilities. Here we chose two different models. First, the "anshpc" tokens and second, the "pack" tokens (see Table 1: for details). If "anshpc" tokens are used, the situation is equivalent to the situation described for Abaqus. If, however, "pack" tokens are in use, one wants to have CPUs containing multiples of eight cores. The reason is that one "pack" token can be used for up to eight CPU cores. If the CPU contains ten cores, two cores are unused or an additional token has to be used. Both situations are inefficient. If two "pack" tokens are available, a single job can use up to 32 cores. Again, having CPUs with multiples of eight cores provides maximum efficiency.

6.2 Using underloading of hardware to maximize cost efficiency

All modern server CPUs use an increased clock rate when less than all cores of a CPU are active at once. As an example, the turbo frequency steps for an Intel Xeon E5-2667v2 CPU are shown in Table 3.
Since hardware costs are comparatively low there is the opportunity to use only a fraction of the capacity of each CPU. That way the clock rate of each used core can be increased.

Number of simultaneously used cores	Max. clock rate per core in MHz
1	3700
2	3600
3	3500
4	3400
5 – 8	3300

Table 3: Overview of turbo frequencies for different numbers of simultaneously used cores in an Intel Xeon E5-2667v2 processing unit.

Usually all CPU cores of a compute server are utilized simultaneously by batch jobs. Since, however, some CAE solvers like LS-Dyna benefit from high per core clock rates, it is worth consider using less than the full capacity of each CPU.

For the example shown in Fig.8: we used a Dell PowerEdge server with two Intel Xeon E5-2667v2 CPUs. Each CPU consists of eight cores, leading to a total of 16 cores. Fig.8: shows the total costs for two different utilization models. The blue line represents results which are obtained by running each job on a machine which is always fully occupied, i.e. all CPU cores are used. The red line represents results for jobs which are run exclusively on one machine. The total costs are different since the hardware costs are not dependent on the number of cores and thus are comparatively high when using small core numbers. The two points at 16 cores differ only within statistical deviations.

The advantage of using exclusive machines is that the total run time of the test job is significantly decreased due to the higher turbo frequency. Considering the costs for a job run with eight cores leads to almost equal costs, although the costs for the hardware are twice as high when running the job exclusively.
In fact, using this particular test job we could not lower the total costs by hardware underloading. However, the difference is very small. It is highly probable that for some jobs hardware

underloading is indeed a possibility to increase efficiency by reducing the total costs for a single CAE job.

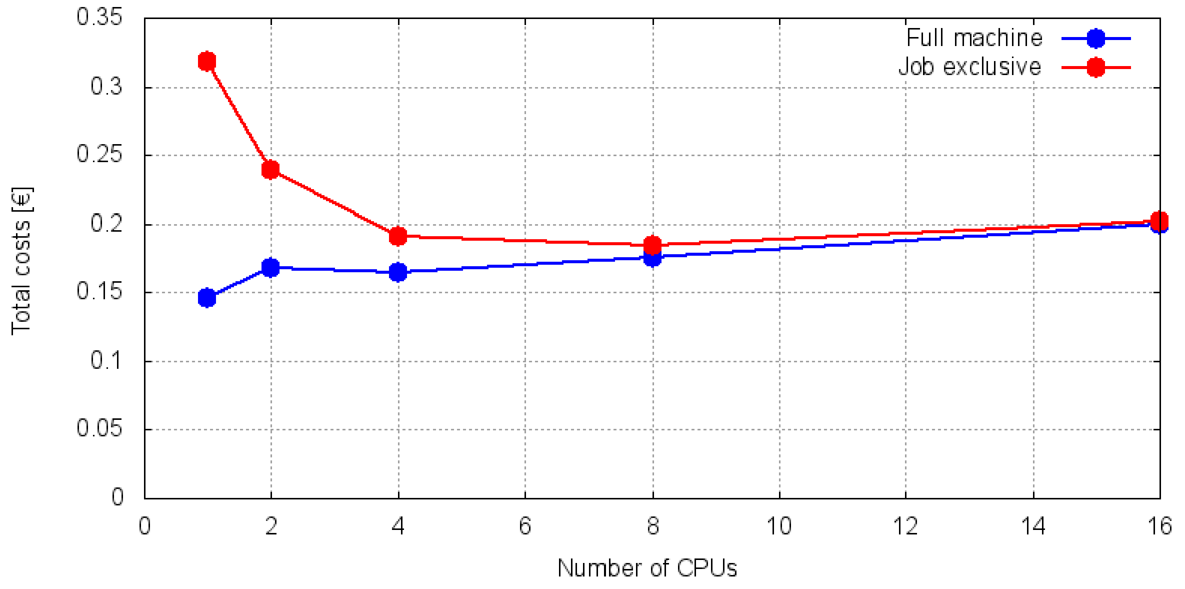

Fig.8: Total costs for a CAE job using two different utilization models. The red line shows the costs for jobs which are run exclusively on one machine while the blue line shows the costs for a job which is part of a completely full machine. The two points at 16 cores differ only within statistical deviations and must be considered equal.

In addition to the higher turbo frequency, we expect an increase in memory performance for less than completely loaded CPUs, which is attributed to the NUMA architecture of modern server CPUs. Especially for implicit jobs which are typically memory-demanding, it might be worth considering hardware underloading.

7 Summary

We used straightforward theoretical considerations in combination with benchmarks for CAE solver jobs in order to derive the total costs of a single CAE solver run. We chose three typical CAE solvers (LS-Dyna, Abaqus, Ansys) to present methods on how the efficiency of small to medium-sized CAE clusters can be maximized. The most important results are:
- License costs are higher by roughly a factor of ten than hardware costs
- Using a CPU which fits well to the license model of the solver used, can increase efficiency.
- Under certain conditions, underloading of hardware could be an opportunity to increase efficiency as well and is certainly worthwhile investigating for your specific jobs.

8 References

[1] „Top Crunch,“ 15 04 2015. [Online]. Available: http://www.topcrunch.org.