Introducing Statistical Design of Experiments to SPARQL Endpoint Evaluation

Kjetil Kjernsmo[1] and John S. Tyssedal[2]

[1] Department of Informatics, Postboks 1080 Blindern, N-0316 Oslo, Norway
kjekje@ifi.uio.no
[2] Department of Mathematical Sciences, Norwegian University of Science and Technology, N-7491 Trondheim, Norway john.tyssedal@math.ntnu.no

Abstract. This paper argues that the common practice of benchmarking is inadequate as a scientific evaluation methodology. It further attempts to introduce the empirical tradition of the physical sciences by using techniques from Statistical Design of Experiments applied to the example of SPARQL endpoint performance evaluation. It does so by studying full as well as fractional factorial experiments designed to evaluate an assertion that some change introduced in a system has improved performance. This paper does not present a finished experimental design, rather its main focus is didactical, to shift the focus of the community away from benchmarking towards higher scientific rigor.

1 Introduction

The practice of benchmarking is widespread, both in industry to select the most appropriate implementation among several choices, and in academia as means to verify or refute an assertion on the performance of a certain practice or system. While we could contribute to the industrial cases, our focus is on the latter. A benchmark typically consists of several microbenchmarks, which are procedures for placing loads on the system to measure its performance as a function of the load. For SPARQL benchmarks, usually the performance is measured as the number of queries the SPARQL implementation is able to answer in a certain time (the throughput) or the time required for it to respond to a query.

However, there are many problems with this approach when used to evaluate assertions about the overall performance of a system or several systems:

- As the complexity increases, the number of parameters that one may wish to test increases dramatically.
- Nothing compels anyone to find microbenchmarks that can refute an assertion, and there is no structured approach to investigate whether a benchmark is flawed.
- There is no meaningful summary of the performance as a whole, so that an assertion about performance cannot be refuted as an optimization may have detrimental side effects that cannot be identified, or cannot be compared to possibly positive effects in a different microbenchmark.

To address some of these problems, one may try to standardize benchmarks, to eliminate some parameters and identify others as believed to be important. Also, social pressure should then prompt developers to use a standard benchmark so that assertions may be refuted.

While this is an improvement which is possibly sufficient for industrial uses, it is still flawed as empirical research, as it cannot capture the full diversity of the problem: For example, if a vendor claims that their implementation is better if used on a highly capable hardware platform, then the standardization of a hardware platform cannot test this assertion. Moreover, to test this assertion, one would need a summary technique that can identify the interaction between implementation and hardware platform and possibly other parameters.

One important aspect of science is the structured approach to falsifying hypotheses, and therefore is important that assertions about performance are formulated as hypotheses that can be falsified.

Therefore, attacking the complexity problem by eliminating parameters we think influence the outcome (e.g. caching) for simplicity is untenable as scientific practice. We must instead tackle this problem by enhancing our capability of testing complex systems and to identify situations where our assumptions are flawed. Such constraints can only be made when we are highly confident that a certain parameter does not have any influence.

In this paper, we employ techniques from the well-established field in statistics known as Design of Experiments (DoE) to point out a direction that promises to address these problems. First, we design a simple 8-parameter experiment, and then we detail an experiment that can meaningfully provide a summary of the statistics.

Then, we demonstrate how the number of runs required in the experiment can be reduced while maintaining a structured approach to the problem, and we discuss the trade-offs involved. We then continue to discuss techniques that show our simplistic experiment to be flawed, yet scientifically better than the practice of benchmarking. Finally, we outline a road-map to establish the use of these techniques in SPARQL endpoint evaluations both as a practical tool and as a scientifically more rigorous methodology.

1.1 Key concepts of Design of Experiments

We expect DoE to be new to most readers, but some familiarity with hypothesis testing is assumed.

In the experiments DoE is concerned with, a *response variable* is measured under a various combinations of parameters. These parameters are known as *factors*. For each factor, a range of possible values is fixed. These values are known as *levels* for this factor. Levels are not constrained to be continuous variables, they can be discrete, or even two different instances of a class. Experiments are run by choosing a combination of levels for the factors.

For example, one may measure the execution time of a SPARQL endpoint by measuring it when it contains 1 or 2 million triples. Then, the execution time is the response variable, the factor is the number of triples and it has two levels, 1

or 2 million triples. We have chosen in this paper to only deal with the relatively straightforward aspects of the DoE formalism, and therefore constrained the experiment strictly to two levels.

In an experiment, there are several factors, and one may run the experiment by measuring the response variable for every combination of levels. In a two-level experiment, this will result in 2^n runs, where n is the number of factors. This is called a *full factorial experiment*. This is done in Section 3.2.

In many fields of science, experimental economy is extremely important, and so, DoE offers extensive methodology to run a certain fraction of the runs at the price of explanatory power. Such experiments are called *fractional factorial experiments* and are covered in Section 3.3.

We describe the influence of the choice of levels on the response in terms of *effects*. For a factor A with two levels, we let a_1 and a_2 be the average response of all 2^{n-1} measurements with A at level 1, and 2, respectively. The main effect of A is then defined as $a_2 - a_1$.

Similarly, *interaction effects* for two factors AB are defined by comparing averages for equal versus non-equal levels of A and B. Details of this theory, and how to compute the effects in practice, using linear regression, are found in [**?**].

The next step is to understand which factors are important or significant. One approach is to plot the sorted effects against the normal distribution. To understand why, note that if there is nothing of interest, the measurements have a certain normal distribution purely due to noise. When plotted against a normal distribution, the plot would be a straight line. If there are any deviations from the straight line, it implies that they may be significant. Such plots can be found in Figs. 1 to 4. The range in the y-axis is dictated by the number of runs.

2 Related work

A literature survey has not revealed any direct prior art, neither in the Semantic Web field nor more generally in database research. However, the general approach has been well established, not only in statistics. A relatively well cited textbook is [**?**] but we have not found the parts discussed in the present paper to be widely adopted. Recently, a comprehensive text on experimental methods has been published in [**?**] demonstrating the broad applicability of the methodology we employ. We have chosen SPARQL Endpoint evaluation as an example in this study since SPARQL Federation is our main interest, and to turn to statistical standard texts [**?**] for this study.

Some problematic sides of benchmarking have been noted in several papers, notably [**?**] and [**?**]. We also acknowledge that great progress has been made to improve benchmarks, in particular, we are indebted to [**?**].

We believe that automatic creation of benchmark queries, as pioneered by [**?**] is a critical ingredient for the application of DoE to be successful as a methodology.

Finally, we acknowledge the efforts of the Linked Data Benchmark Council and the SEALS project. However, they appear to focus on cases where the workload is assumed to be well characterized, rather than scientific evaluations.

3 Experiments

Common experiments include the comparison of several different implementations. However, we focus on a slightly different problem: We would like to compare the same implementation before and after some change has been made, typically with the intention to enhance the performance of the system. There are two reasons for this: One is that there are currently many SPARQL implementations available, also Free Software ones that can be modified by anyone, and the other is to constrain the scope of the study to comparing just two different things.

3.1 Setup

As the focus of this paper is didactical, the actual measurements are much simpler than those that have been used in benchmarking. We have used the data-set of DBPedia SPARQL Benchmark [**?**], but only consider the smallest of their data-sets (that we found to be more than 15 MTriples). Moreover, we have taken subsets of that data-set by using the last 1 or 2 MTriples from their file.

We have chosen 8 factors each having 2 levels, "TripleC" the number of triples in the data-set, 1 or 2 MTriples, "Machine", which is the software and hardware platform, one larger with slower disks[3] and one smaller with a faster disk[4].

Then, we test some language features: "BGPComp", which is a Basic Graph Pattern of varying complexity.

Level 1	**Level 2**

`?s  rdfs:label ?l1 ;`
`  ?p1 ?o1 .`
`?o1 dbo:populationTotal ?l2 .`
	`?s  rdfs:label ?l1 ;`
	`  ?p1 ?o1 .`
	`?o1 dbo:populationTotal ?l2 .`
	`?s foaf:page ?o2 ;`
	`  dbpprop:subdivisionName ?o3 .`
	`?o3 skos:subject ?o4 ;`
	`  dbpprop:seat ?o5 ;`
	`  a ?c1 .`

The following factors test the absence (at level 1) or presence (at level 2) of the following clauses:

[3] Running GNU/Linux Debian Squeeze, has 16 GB RAM, an Intel Core2 Duo E8500 CPU and two Parallel-ATA disks in a RAID-1 configuration

[4] Running Debian Wheezy, has 8 GB RAM, an Intel Core i7-3520M CPU and a single SSD on SATA-III

"Union"	**"Lang"**
`{ ?o1 dbpprop:longd ?long ; dbpprop:latd ?lat . } UNION { ?o1 geo:long ?long ; geo:lat ?lat . }`	`FILTER langMatches( lang(?l1), "fr" )`
"Range"	**"Optional"**
`FILTER (?l2 > 800000)`	`OPTIONAL { ?o1 foaf:homepage ?o6 . }`

These fragments have been carefully designed for illustrative utility, as well as suitable selectivity. They are not themselves important, they only serve to illustrate certain examples of possible factors. When the experiment is run, they are combined to yield a query, which is then sent to a SPARQL endpoint. The SPARQL endpoint is itself set up using 4store (see [**?**]) version 1.1.5.

Finally, "Implement" is the implementation undergoing evaluation. Level 1 is running the new implementation, whereas Level 2 is the old implementation. The null hypothesis H_0 is that the old implementation is as good as the new, and the alternative hypothesis H_1 is that the new implementation has overall improved the performance of the system.

A real optimization is beyond the scope of this paper. Instead we have performed a simulation that enables us to understand the effect of the changes. In a real-world case, it is often the case that an optimization has negative side-effects, and we would need to simulate both the optimization and the side-effects. To do that, we degraded the performance of the 4store SPARQL implementation by inserting `sleep` statements. Specifically we inserted the C statement `sleep(2)` on line 920 in `src/frontend/query.c` on level 1, to simulate the optimization. This has the effect of delaying execution for 2 seconds for every block for all kinds of joins. On level 2, we inserted `usleep(2000)` on line 987 in `src/frontend/filter.c` to simulate the negative side-effect. This delays execution for 2 milliseconds every time the `langMatches` SPARQL function is called.

The experimentation is implemented in R [**?**], which is a free software environment for statistical computing and graphics. Necessary tools for DoE has been implemented by the R community in packages called DoE.base [**?**] and FrF2 [**?**]. The experiments are run on a third computer in a 1 gigabit Ethernet LAN with the two experiment hosts. On the experiment hosts, four 4store instances runs sequentially and independently on different ports, where processes that are not actively answering a query are idle. Practically, the "Machine" factor specifies a hostname, whereas "TripleC" and "Implement" specifies a port number.

As response variable, we have chosen to use the time from the namespace server lookup finishes to the data has been transferred. This choice is somewhat arbitrary, many other response variables could be chosen, and indeed, future work should explore multi-variable responses, but for simplicity, we think this

reflects the total amount of work done by the SPARQL endpoint well. For the measurements, we have chosen to use the RCurl package [**?**], which is a thin wrapper around the curl library. Curl has well developed facilities for timing requests and responses, and so we rely on its measurements.

Finally, note that there are two common issues in benchmarking we do not consider: we do not allow a warm-up run, nor do we take into account that the server may cache all or parts of the result set. The reasons for this choice will be discussed later.

All the code to reproduce this study as well as detailed instructions have been published on Github: https://github.com/kjetilk/doe-sparql.

3.2 Full factorial experiment

In the full factorial experiment, all combinations of the 8 factors are executed, in total 256 runs. This is called a 2^8 factorial experiment, and while it quickly becomes infeasible for many factors, we shall nevertheless find the following small example instructive. Note that each combination is executed only once (i.e. it is *unreplicated*), however, so even a full factorial experiment compares well to a typical benchmark in which each microbenchmark must be executed several times to obtain enough data to compute a mean with reasonable certainty.

The above experimental setup is executed across a design matrix generated with the DoE.base package, which returns a data frame consisting of the factors with a column containing the corresponding curl measurements. With that, we do as described in the introduction and fit a linear model with all factors. Then, we generate a normal plot, see Figure 1. Any significant departure from a straight line can be interpreted as a significant effect (see Section 4.8 of [**?**] for a detailed explanation). In our case, it is most important to note that any effect that is negative means the effect enhances the performance of the SPARQL endpoint, the runtime decreases. If "Implement" and its interactions are negative conditional on that the factors involved with its interactions are on their high level, then it supports the alternative hypothesis, i.e. we have successfully improved the performance.

To proceed beyond a visual judgment of significance, we may use Lenth's method [**?**], which is a simple method for estimating significance in an experiment with no replicates. In Table 1, the most important effects are listed. In Figure 1, Lenth's method is used to label the significant effects. We see that we have many very significant effects (unfortunately so many that some labels are illegible). We also note that "Implement" itself is significantly negative and that the "Implement:Lang" interaction is significantly positive, as expected from our setup, where "Lang" was the detrimental side-effect we simulated.

To investigate whether the detrimental side-effects cancel the positive effect of our simulated optimization, we have to formulate a hypothesis test. By inspecting the normal plot in Figure 1 and Table 1, we see that six factors are highly significant by either being a significant main effect or participate in a highly significant interaction. We say that the two factors "Range" and "Machine" are *inactive* since they do not contribute significantly to the observed variation in

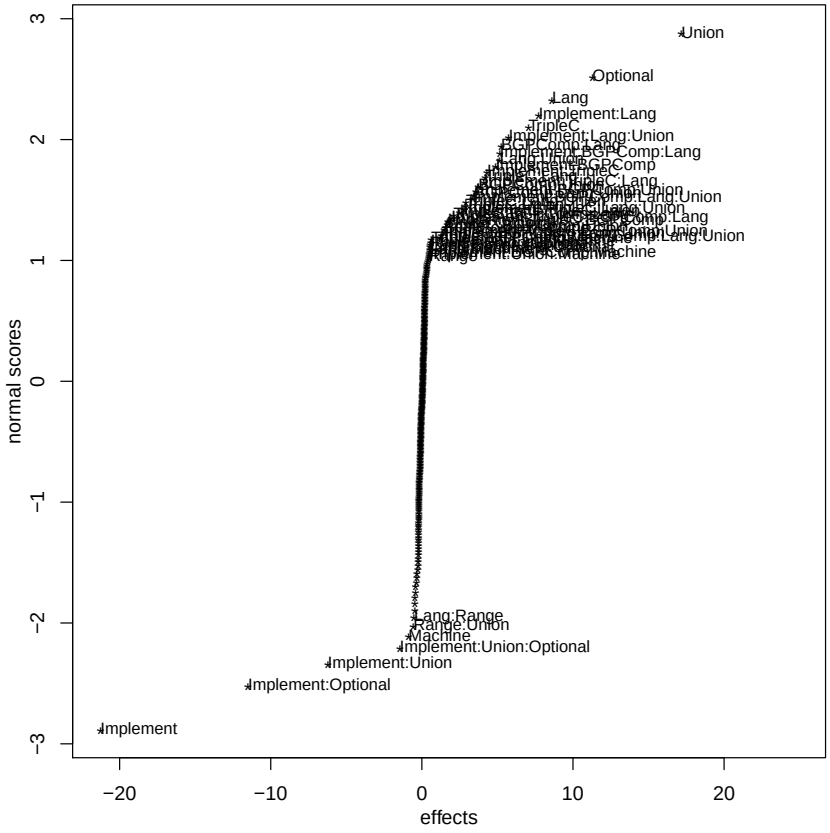

Fig. 1. A normal plot of the Full Factorial Experiment. The labelled points are considered significant by using the Lenth criterion at a level $\alpha = 0.05$.

performance. "Machine" is marginally significant, but the effect is so little, it will be of greater use for us in the following procedure. In practical situations, the five factors "BGPComp", "Lang", "Optional", "Union" and "TripleC" can be regarded as given, i.e. we cannot change the size of the data-set without loss of information, or change the query, since it gives a different answer. In practice, we can only control "Implement", i.e. we can only change the implementation. In this context, we call "Implement" a *control* factor and the other five active factors *environmental*.

We would like an overall test to see whether the new implementation is better than the old, and the presence of inactive factors makes it possible to average the performance of the new and the old implementations into two distinct vectors. We do this by creating an average over all the 32 level-combinations of the five active environmental factors. This leaves us with 4 values for each of the two levels of "Implement" and a two-sample t-test can be performed. Effectively, we treat our experiment as a 2^6 experiment replicated 4 times.

Table 1. The magnitude of effects for some important main effects and interactions.

Factors	Effect
Implement	−21.27
Implement:Optional	−11.49
Implement:Union	−6.21
Implement:TripleC:Lang1	4.12
TripleC:Lang	4.20
Implement:TripleC	4.32
Implement:BGPComp	4.89
Lang:Union	5.16
Implement:BGPComp:Lang	5.16
BGPComp:Lang	5.25
Implement:Lang:Union	5.75
TripleC	7.06
Implement:Lang	7.73
Lang	8.62
Optional	11.30
Union	17.18

We perform the test as a one-sided hypothesis test where H_0 is that the mean of the vectors are equal and H_1 is that mean sof the vectors representing the new implementation is lower. In this example, we find that the available data supports the assertion that the new implementation is better with a high probability, $p = 1.16 \cdot 10^{-07}$.

While it is interesting to test the overall hypothesis that the new implementation is an improvement, it is also interesting to know if there are cases where it fails. Again, we treat the experiment as a 2^6 experiment replicated 4 times. Now, for each of the 32 resulting combinations of the environmental factors for each level of "Implement", we extract 4 values. We may now run a t-test for each of the replications, as above. The results are tabulated in Table 2. We see that there are significant improvements in most cases; one may want to further investigate the cases where they do not.

For future larger experiments, one must investigate whether it is necessary to adjust the p-value due to the larger numbers of hypotheses tested.

The main objective of this paper is not to establish that the optimization is significant, it is merely a simulation. It is to establish a critical practice of evaluations. We must therefore understand why some effects are significant. That "Implement" is significant is hardly a surprise, that is what we worked for. That the main effects "Union", "Optional" and "Lang" are strong is also due to that these are fairly demanding things to evaluate. Since the delay we inserted affects all kinds of joins, it is also intuitive that interactions between "Implement" and those that require joins are strong. The strong "BGPComp:Lang" and "Lang:Union" interactions might be due to that many more triples need to be searched for a language tag in one of the levels of the interacting factors. However, the positive "Implement:TripleC" interaction evades such explanations; in fact, it hints that our optimization may not work as well for larger databases. By carefully inspecting Table 2, we see that whenever "TripleC" is at the low level, the new implementation is always best. Only at the higher level, the old implementation may be better. This is a clear indication that the separation between levels for the sizes of the data-set is too small and should be investigated further. Such reasoning should be applied to all significant effects.

Table 2. p-values for different parts of the experiment

"TripleC"	"BGPComp"	"Lang"	"Union"	"Optional"	p
1	1	1	1	1	0.012
1	1	1	1	2	$1.4 \cdot 10^{-09}$
1	1	1	2	1	$3.1 \cdot 10^{-09}$
1	1	1	2	2	$6.1 \cdot 10^{-11}$
1	1	2	1	1	$3.3 \cdot 10^{-06}$
1	1	2	1	2	$2.7 \cdot 10^{-09}$
1	1	2	2	1	$2 \cdot 10^{-06}$
1	1	2	2	2	$3.6 \cdot 10^{-10}$
1	2	1	1	1	0.014
1	2	1	1	2	$1.2 \cdot 10^{-10}$
1	2	1	2	1	$2.8 \cdot 10^{-14}$
1	2	1	2	2	$4.1 \cdot 10^{-15}$
1	2	2	1	1	$2.1 \cdot 10^{-05}$
1	2	2	1	2	$2.7 \cdot 10^{-07}$
1	2	2	2	1	0.0072
1	2	2	2	2	$1.6 \cdot 10^{-05}$
2	1	1	1	1	0.28
2	1	1	1	2	$3 \cdot 10^{-07}$
2	1	1	2	1	$3.3 \cdot 10^{-07}$
2	1	1	2	2	$1.7 \cdot 10^{-08}$
2	1	2	1	1	0.0023
2	1	2	1	2	$6.5 \cdot 10^{-07}$
2	1	2	2	1	0.00032
2	1	2	2	2	$1.3 \cdot 10^{-06}$
2	2	1	1	1	0.013
2	2	1	1	2	$1 \cdot 10^{-11}$
2	2	1	2	1	$3.8 \cdot 10^{-11}$
2	2	1	2	2	$4.1 \cdot 10^{-15}$
2	2	2	1	1	1
2	2	2	1	2	$2.3 \cdot 10^{-05}$
2	2	2	2	1	1
2	2	2	2	2	0.99

3.3 Fractional factorial experiment

As mentioned, the full factorial experiment goes as 2^n and becomes prohibitively expensive when n is a large number of factors. As the evaluation of SPARQL endpoints is an inherently complex problem, a large number of factors are needed, and so full experiments cannot scale. However, we may reduce the size of the experiment significantly by sacrificing some explanatory power, in the form of fractional factorial experiments.

We lose explanatory power due to *aliasing*; for example, the interaction labelled "TripleC:BGPComp" may be aliased with the "Machine:Range" interaction. That is to say, a detected increase in run time from larger Basic Graph

Patterns for large databases, can also be explained by a less powerful machine that is worse at evaluating FILTER clauses with ranges. They are indistinguishable. This may or may not cause a problem. In many cases, we may not be interested in these effects, we may only be interested in "Implement" and its interactions. Moreover, it is possible (even easy using the FrF2 package in R) to declare which effects must not be aliased, and determine the size of the experiment based on that. Such a main effect or two-factor interaction is called *clear* if none of its aliases are other main effects or two-factor interactions. Another possibility to shrink the size of the experiment is if we *a priori* can say that some effects are negligible. We have not found such assumptions to be tenable in our case.

We have made two fractional factorial designs: One with 32 runs and one with 64 runs. Again, the number of runs is in powers of 2, but much smaller. In both cases, we have specified that all the factors must be clear and also all two-factor interactions where "Implement" is involved must be clear. The resulting experiments is returned to us as design matrices, but they can also be described sufficiently for reproducibility in terms of *design generators*. These also declare aliasing relations for some of the main effects. The design generators are

"Range" = "TripleC" "Machine" "BGPComp"
"Union" = "TripleC" "Machine" "Lang"
"Optional" = "TripleC" "BGPComp" "Lang" "Implement"

and

"Union" = "Implement" "TripleC" "Machine" "BGPComp"
"Optional" = "Implement" "TripleC" "Lang" "Range"

for the 32 and 64 run experiments respectively.

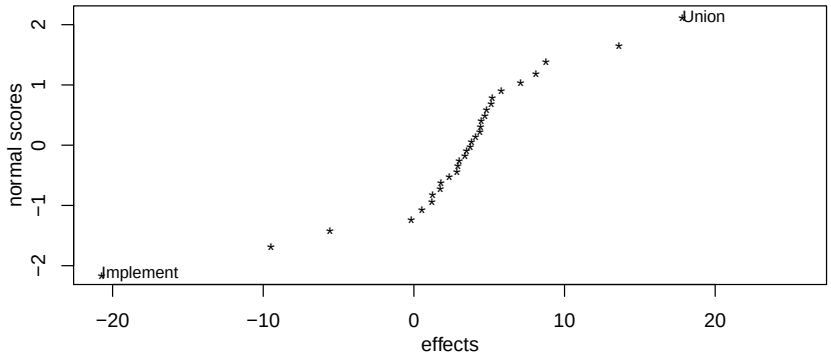

Fig. 2. A normal plot of the Fractional Factorial Experiment with 32 runs. The labelled points are considered significant by using the Lenth criterion at a level $\alpha = 0.05$.

The resulting normal plot is in Figure 2. We see that only two main effects are significant on a $\alpha = 0.05$ level according to the Lenth criterion, "Implement" is again deemed a significant improvement, and "Union" is deemed significantly

the hardest operation for the query engine. We see that some other points also deviate from the straight line, but with as few runs as this, the total variance is great, so no other conclusions can be drawn. We may use the t-test we used in the previous section, as we have only two active factors and so we may treat it as a 2^2 experiment with 8 replications. The resulting $p = 0.00098$ is still very low, albeit larger than in the previous experiment. This also explains why there are so few significant effects, and had to be expected with such a small number of runs and thus higher variance.

For any other purpose than a rough idea of some key influences on the overall performance of the endpoint for the given levels, this little experiment is insufficient.

We turn to the 64-run experiment and its normal plot in Figure 3. We see that then we get more significant effects, and that they correspond well to those we saw in the full factorial experiment. They also all have an intuitive explanation as above. However, we do not see the effects that we found worrisome in the full factorial design. The "Implement:TripleC" interaction emerges only for a level $\alpha = 0.15$. To ensure that we discover such cases will be a key challenge in further studies.

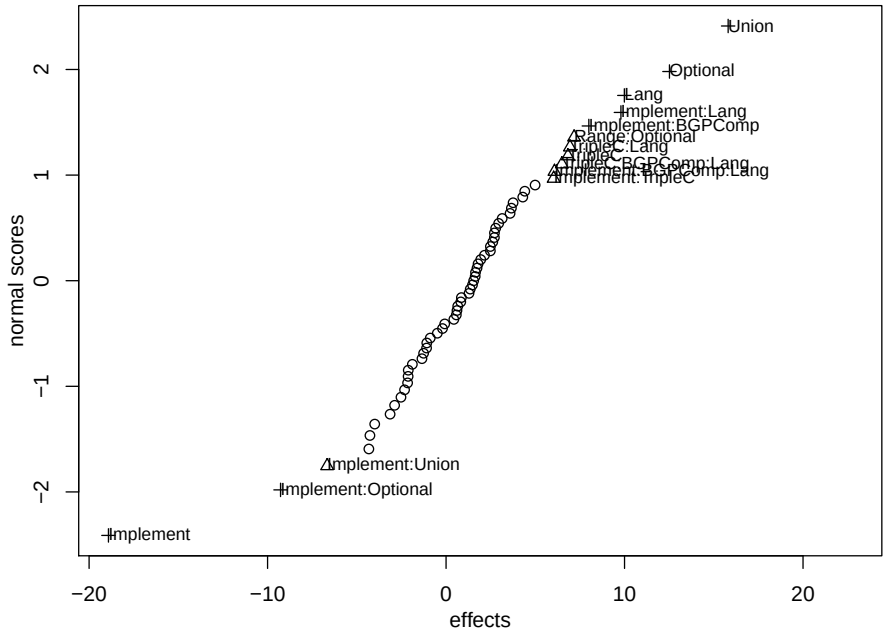

Fig. 3. A normal plot of the Fractional Factorial Experiment with 64 runs. The labelled points are considered significant by using the Lenth criterion. Triangles correspond to a level $\alpha = 0.15$ and crosses at $\alpha = 0.05$. The use of different symbols has been added to FrF2 by the authors.

Since the 64-run experiment has 3 inactive factors, "Machine", "TripleC" and "Range", we can treat it as a 2^5 experiment replicated twice. Using the same procedure as above, we find that again, the hypothesis test indicates that the new implementation is an improvement at $p = 5.9 \cdot 10^{-6}$. Again, the comparatively higher p-value is due to the lower number of runs. It is also possible to perform the per-environmental factors test.

3.4 Fractional factorial experiment with more data

The importance of finding flaws in the levels is highlighted by the normal plot in Figure 4. The smaller experiment made it feasible to run with a larger data-set, this experiment uses the original 15 MTriples data-set of [**?**]. The picture is now as hinted in the previous discussion, for larger "TripleC", our simulated performance is actually worse, as evident from the positive "Implement". We also see that the interactions "Implement:Lang" and "Implement:TripleC" are highly significant. It is clear what happened: For larger "TripleC", the delay we introduced in the langMatches function has a devastating effect, since it delays the execution for every triple that the query engine applies the filter to, and thereby completely dominates the execution time.

Moreover, we note that the "Machine" factor has become highly significant in the other end along with its "Machine:TripleC" interaction. One should take careful notice that the "Machine" factor encompasses many different things, that should be different factors; CPU, disks, available RAM, installed software, tuning parameters and so on. It is advantageous to have such broad factors in the design in early experiments, because they may point out unexpected effects that should be further scrutinized. The "Machine" factor is a good example of usage that also helps us manage complexity: If possible, one may group many factors into one if in doubt of their relevance and break it up in subsequent experiments if deeper understanding is needed. To some extent, "BGPComp" is also a broad factor, since it encompasses many structures that are normally seen in a Basic Graph Pattern. In this case, the cause of the performance problem was highly audible: It was the old P-ATA disks of the larger machine, but in general, such clues are not available to the experimenter, therefore more factors are needed.

We are forced to conclude that the three preceding experiments are severely flawed, as they failed to take into account the effects from larger data sizes on slow hardware.

Note that we do not take advantage of the usual performance of 4store in this simulation, to the contrary, we chose 4store because of its relative simplicity to modify it to suit our needs for illustrative purposes, which consists of degrading it in several ways. The small data-sets used here do not imply that the approach will not work for large data-sets; properly configured, actual optimization will give very different runtimes, and the small number of runs will make it possible to evaluate a very large number of factors.

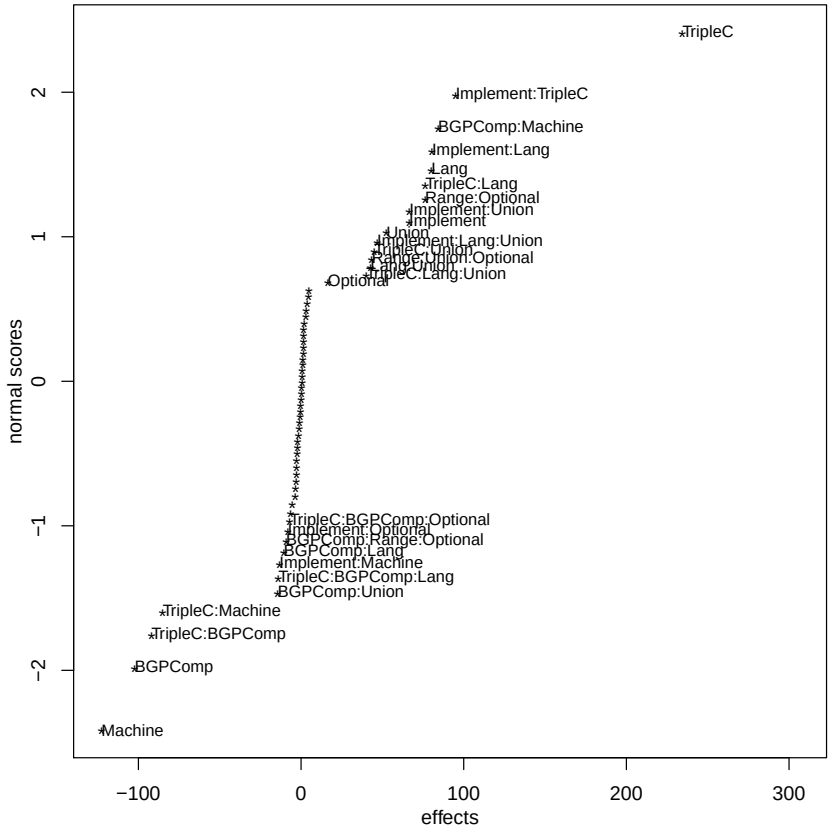

Fig. 4. A normal plot of the Fractional Factorial Experiment with 64 runs where the level 2 of "TripleC" is 15 MTriples. The labelled points are considered significant by using the Lenth criterion at a level $\alpha = 0.05$.

4 Discussion

It is a fundamental property of two-level experiments that they can only model linear effects, however, they are usually quite sufficient for identifying significant effects even though the underlying response may be non-linear. As we saw in Section 3.2, the choice of levels may be critical, and if non-linearity is expected, more than two levels must be considered, or at the very least, attention must be paid to the choice of levels.

We have argued that fixing certain parameters is untenable, but we have allowed ourselves to disregard caching and warm-up runs. We are aware that caching may be important in 4store, as a preliminary experiment gave a response time of 53.9 s of a query with "BGPComp" at level 2 but other factors at level 1. When an OPTIONAL clause was added, the response time dropped to 1.25 s. As this is not a more restrictive query, the most likely explanation is that the

result of the evaluation of the Basic Graph Pattern was cached, so only the OPTIONAL left join was necessary to add in the second query.

We did this to illustrate *randomization* as this is what makes it permissible, i.e. the order of execution is random, so the benefit of caching is likely to apply randomly to different runs. For example, had the run order been reversed in the previous example, it could have been the query without OPTIONAL that would have benefited. Unless the effect of caching is cumulative throughout the experiment (which is possible), the randomization will have the effect of reversing the runs randomly. The end result is that the effect of so-called "lurking variables" such as caching, warm-up effects, etc, contribute to the total unexplained variance of the experiment, but should not skew the results to invalidate the estimated effects.

As the unexplained variance of the experiment may cloud important effects, notably effects that indicate flaws, it should be kept to a minimum. Thus, lurking variables should be turned into factors whenever possible, as they degrade the quality of the experiment, but in many cases, we can live with them if their contribution to the variance is small.

Finally, we note that the most problematic cases are those that are covered neither by the broad factors, nor the lurking variables, or any of the specific factors. We cannot in the general case be sure that they are accounted for, so reviewers must remain vigilant that some factors are simply ignored. What finally invalidates the present experiment is the complete absence of certain obvious language features, such as solution modifiers.

5 Future Work

The present experiment is very primitive in its choice of factors, and we believe that the strategy employed to construct suitable experiments will be the main determinant for the long-term feasibility of this direction. One strategy will be to start with broad factors such as "Machine" and refine them with experience. Also, the data must be parameterized with many more factors than just the number of triples; data heterogeneity [**?**] is one example, but also queries over highly skewed data is important. A more difficult issue is how to address the problem of testing the whole SPARQL language [**?**]. We believe that the parameterization work initiated by [**?**] with their SPLODGE system is a key ingredient. They parameterized SPARQL queries in such a way they could be auto-generated. One suggestion is to parameterize queries based on the grammar, so that one factor becomes the number of e.g. UNIONs, and the levels are chosen. One problem one will soon run into is that parts of the language is recursive, e.g. a GroupGraphPattern can consist of several GroupGraphPatterns. However, we believe that pragmatic limits can be set, it is for example not of practical interest to nest OPTIONALs to any great depth even if it is allowed by the language.

We saw in Section 3.3 that with a smaller experiment, fewer interactions are significant, and so provide us with fewer clues to assess the soundness of the experiment. This should be a focus of further research.

In this paper, we have only employed the simplest parts of the DoE theory. What has been presented here is part of a much more general formalism known as Orthogonal Arrays. The use of orthogonal arrays allow for different numbers of levels, for much greater flexibility in total run size, and for non-regular designs that can provide a good fit for the complex problem of SPARQL endpoint evaluations.

Finally, as each run is cheap, and the experiment can usually run without human intervention, we believe it is interesting as a case for advancing the state of the art of DoE.

6 Conclusions

As evaluating SPARQL endpoints is inherently difficult, a simplistic experiment such as the one we designed should not hold up to scrutiny. We set out to demonstrate how an experiment could be set up using DoE, and show how it can be analyzed. In sections 3.2 and 3.3, we first saw how the analysis correctly pointed out the most important effects, under assumptions dictated by the factors and levels that were given. We saw how the formalism provided a comprehensive view of the experiment. Then, we saw that the formalism pointed out weaknesses that could invalidate the experiment, and in Section 3.4 we saw that the experiment did indeed not hold up to scrutiny.

We saw that while two-level experiments can perform well in determining significant effects, they do not necessarily work well to find a model, as exemplified by failure to identify the possibly non-linear effect of the P-ATA disks in the first experiments. This issue can be addressed by using an orthogonal array design.

Also, experiments as small as the 32-run are not useful in estimating any detailed characteristics and are thus of little use for science, but could be useful in engineering: If the experiment has been validated by a larger experiment, it could serve well in a Continuous Integration system.

We have seen that we can perform a proper hypothesis test based on summary statistics, and by using our expertise, we showed how to reveal that the experiment was flawed, and we have cautioned that failure to control unexplained variance may compromise our ability to do so. We have also pointed out how this direction can provide more rigorous evaluation practices.

To this end, the following questions must be asked:

1. Are there factors that cover all realistic features?
2. If not, are they adequately covered by randomization?
3. If so, would the variance resulting from randomization obscure factors that could provide clues that the levels are wrongly set?
4. By carefully examining interactions with “Implement”, are there any that are unaccounted for, and that could point out wrongly set levels?

Even if the complexity is great, with properly tuned endpoints, it is feasible to do millions of runs, and with orthogonal arrays, this formalism can be extended to many different evaluation problems.

Acknowledgments Kjetil Kjernsmo would like to thank his main supervisor Martin Giese for kind assistance. He would also like to thank Steve Harris for his generous support in degrading his excellent work on 4store.