iTuned: A Tool for Configuring and Visualizing Database Parameters

Vamsidhar Thummala
Duke University
Durham, NC
vamsi@cs.duke.edu

Shivnath Babu*
Duke University
Durham, NC
shivnath@cs.duke.edu

ABSTRACT

iTuned is a tool that takes a SQL workload as input and recommends good settings for database configuration parameters such as buffer pool sizes, multi-programming level, and number of I/O daemons. iTuned also provides response-surface and sensitivity-analysis plots that help database users analyze the impact of each parameter. iTuned has the following novel features: (i) a technique called Adaptive Sampling that proactively brings in appropriate data through planned *experiments* to find high-impact parameters and high-performance parameter settings, (ii) an executor that supports on-line experiments in production database environments through a cycle-stealing paradigm that places minimal overhead on the production workload, and (iii) portability across different database systems. This demonstration will focus on the interesting use-case scenarios of iTuned, and show its effectiveness on recommending configuration settings for popular database benchmarks.

Categories and Subject Descriptors

H.3.4 [**Systems and Software**]: Performance evaluation

General Terms

Experimentation, Performance

Keywords

Configuration, Tuning, Visualization

1. INTRODUCTION

Tuning a database system to meet desired performance objectives can be a challenging task. Although most commercial database systems have wizards for tuning the logical and physical database design (e.g., indexes), few practical tools are available for tuning the settings of database configuration parameters. Database systems ship with a large number of parameters like buffer pool sizes, number of I/O daemons, and multi-programming level whose settings affect the database configuration directly; as well as other parameters like cost of a random page fetch, CPU cost of processing a tuple, and OS buffer cache size that impact the query optimizer's decisions during plan selection. The default settings of these parameters are tuned for wide compatibility rather than performance.

The task of configuring the database system with appropriate parameter settings is both challenging and time-consuming. The only tools for parameter tuning that we are aware of are: (i) IBM DB2's *Configuration Advisor* [4], and (ii) PostgreSQL's *pgtune* [5]. Both tools recommend default configuration settings based on answers provided by the Database Administrator (DBA) to high-level questions, and employ heuristics to best reflect popular rules-of-thumb for tuning. For example, pgtune tries to automate the tuning guidelines as specified by the PostgreSQL wiki [6] page.

Good settings for configuration parameters depend on the type and amount of resources available and the type of workload that runs on the system. We have shown in our earlier research [2] that rule-based and default settings may not yield good performance even for simple workloads; and can cause frustrating experiences for users who lack a deep understanding of different parameters. For complex workloads, where inter-parameter interactions can become prominent, rule-based and default settings may degrade performance significantly. Inter-parameter interactions make parameter tuning nontrivial even for experienced DBAs.

DBAs typically follow trial-and-error methods to come up with good parameter settings. This process involves setting up a replica of the production environment on a test system, conducting some test runs with different parameter settings and a representative workload to observe performance, and doing further runs until a setting with the desired performance is obtained. The whole process is manual, labor-intensive, and requires a deep understanding of the database system's internals and behavior so that the time for tuning can be minimized.

This demonstration introduces our *iTuned* tool that automates the process of setting database configuration parameters [2]. iTuned infers good settings efficiently by conducting planned *experiments* to learn the response surface of performance as the parameters are varied. iTuned takes a holistic approach to parameter tuning:

- iTuned's *planner* uses a technique called Adaptive Sampling that selects a schedule of experiments aimed to find high-impact parameters and high-performance parameter settings.
- iTuned's *executor* conducts experiments in production environments through a cycle-stealing paradigm that places minimal overhead on the regular database workload.
- The techniques used by iTuned's planner and executor are not coupled tightly to any database system, so iTuned is portable across database systems.

To our knowledge, iTuned is the first practical tool that uses planned experiments to tune database configuration parameters. Our demonstration will illustrate how iTuned benefits users and DBAs. In the

*Supported by NSF grants IIS 0644106 and IIS 0917062

SIGMOD'10, June 6–11, 2010, Indianapolis, Indiana, USA.

next section, we present a high-level overview of iTuned's usage and architecture. Section 3 describes our demonstration scenarios. We refer the reader to [1, 2] for a detailed description of iTuned's implementation and evaluation.

2. OVERVIEW OF ITUNED

iTuned can be invoked by a DBA whenever the database performance for a given workload is unsatisfactory due to the current setting of configuration parameters.

```
<iTunedConfiguration version="0.3">
  <!-- A plan defines the collection of
       experimental runs -->
  <Plan>
    <!--Repeat can be set to run each
        experiment multiple times -->
    <Run name="ParameterTuning" repeat="1">
      <!-- Choice of experimental design to
           use for first set of samples -->
      <InitialDesign>LatinHypercubeSampling</
          InitialDesign>
      <!-- Adaptive Sampling algo used -->
      <AdaptiveSampling>
          GaussianProcessRegression</
          AdaptiveSampling>
      <!-- Type of executor used for
           experiments-->
      <Executor>iTunedExecutor</Executor>
      <!-- Input specification -->
      <Inputs>
        <!-- Parameters and values/ranges -->
        <Input name="shared_buffers" range="0
            MB-1000MB"/>
        <Input name="work_mem" value="5MB"/>
      </Inputs>
      <!-- Output specification -->
      <Outputs>
        <!--Each output parameter needs an
            accompanying objective or a
            constraint defined -->
        <Output name="Running time" objective=
            "min">
        <Output name="Time budget" constraint=
            "1hr">
      </Outputs>
    </Run>
  </Plan>
</iTunedConfiguration>
```

Listing 1: Example specification file for a tuning session

2.1 Input Specification

DBAs interact with iTuned by specifying inputs using a graphical front-end interface and an XML file, and then visualize the tuning output using iTuned's Visualizer (Section 2.4). An XML template file is supplied with iTuned and is typically configured once per installation. The values to be configured in the XML file include the ranges of the parameters, the type of executor, and the algorithms used for sampling. Through the front-end interface, DBAs can specify the subset of configuration parameters for tuning for the current workload. To start a tuning session, the following should be specified using iTuned's front-end interface:

- List of configuration parameters for (current) tuning
- Database workload of SQL queries
- *Server URL* specifying the database server to be tuned, and *workbench URLs* specifying one or more database servers where experiments can be run
- Time budget for tuning

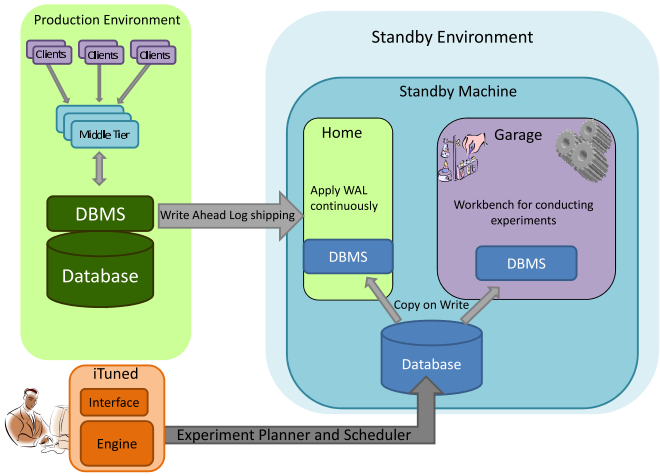

Figure 1: iTuned's executor in action for standby databases

Listing 1 and Figure 2 show an example specification of a tuning session. The list of configuration parameters can be chosen based on the workload characteristics. For example, work_mem (used for in-memory sorts in PostgreSQL) is an important parameter in data warehousing where users submit large queries involving complex joins, group-bys, and order-bys. Similarly, checkpoint-related parameters are important in transactional workloads where there is a trade-off between extra overhead during regular use and the recovery time when a failure happens.

The DBA also provides the database workload as an input to iTuned. The workload can be collected in any number of ways, e.g., by tracing the queries executed on the production server over a period of time. The server URL specifies the database to be tuned, and the workbench URLs (which can include the server URL) specify the databases where iTuned can run experiments. The time budget specifies the maximum time for iTuned to run experiments and produce the tuned configuration settings. Note from Listing 1 that the DBA can specify the experiment-planning algorithm to be used.

2.2 Planner

iTuned's experiment planner determines which experiments to conduct for a given workload, W, within the allocated budget. This problem is related to the sampling problem in databases. We can consider the information about the full response surface S_W to be stored as records in a (large) table T_W with d attributes $x_1, \ldots, x_d$ along with the performance measure y. An example record $\langle x_1 = v_1, \ldots, x_d = v_d, y = p \rangle$ in T_W says that the performance at the setting $\langle x_1 = v_1, \ldots, x_d = v_d \rangle$ is p for the workload W under consideration. Experiment selection is the problem of sampling from this table. However, the difference with respect to conventional sampling is that the table T_W is never fully available. Instead, we have to pay a cost—namely, the cost of running an experiment—in order to sample a record from T_W.

Adaptive Sampling: iTuned uses this technique to plan the experiments to be conducted. Adaptive Sampling analyzes the samples collected so far to understand how the response surface looks like, and where the good settings are likely to be. Based on this analysis, more experiments are done to collect new samples that add maximum utility to the current samples. An initial set of experiments is done to bootstrap the whole process. Adaptive Sampling can be implemented using various algorithms. One such algorithm that can deal with simple to complex response surfaces uses *Gaussian Process Regression* [2, 3]. The initial set of experiments for bootstrapping can be picked using *Latin Hypercube Sampling* [2].

2.3 Executor

iTuned's executor is responsible for actually running the experiments selected by the planner in a production environment. The guiding principle of the executor is: exploit underutilized resources in the production environment for experiments, but never harm the production workload. The design of the executor is based on *control policies* and *resource containers*. The control policy specified with each resource dictates when the resource can be used for experiments. Resource containers split the functionality of each resource into two: (i) *home use*, where the resource is used directly or indirectly to support the production workload, and (ii) *garage use*, where the resource is used to run experiments. Containers are resource-capped and isolated virtual machines. We have implemented resource containers using the *zones* [7] feature in Solaris.

Figure 1 illustrates the home/garage design using the standby database. When the standby machine is not running experiments, (only) the home container runs on the machine using all the available resources. The garage container lies idle and uses zero resources. The garage container will be booted only when the policy kicks in and allows iTuned to schedule an experiment on the standby machine. Once booted, the garage takes a snapshot of the current data from the home container. The garage's snapshot is logically separate from the snapshot used by the home container, but it is physically the same except for *copy-on-write* semantics. iTuned's current implementation of snapshots and copy-on-write semantics leverages the Zettabyte File System (ZFS).

2.4 Visualizer

iTuned's visualizer displays the following sets of plots for a tuning session:

- 2D projection of the performance response surface as illustrated in Figures 2-4. These plots help the DBA to get an intuitive feeling of how performance varies as the parameter settings are changed. For example, a flat response surface plot indicates that the parameters do not impact performance. Furthermore, using these plots, the DBA can narrow down quickly to high-performance regions.
- Sensitivity-analysis plots like Figure 5 that show which parameter settings are most sensitive to the workload performance.

3. DEMONSTRATION PLAN

Our demonstration will show the complexity of configuration parameter tuning, and how iTuned can be used interactively by DBAs for performance tuning. For illustrative purposes in this proposal, we use a simple workload consisting of TPC-H query 18 (Listing 2). The actual demonstration will consider simple and complex database workloads running on open-source PostgreSQL database system consisting of more than 100 configuration parameters.

```
-- TPC-H Large Volume Customer Query (Q18)
SELECT c_name, c_custkey, o_orderkey,
     o_orderdate, o_totalprice, SUM(
         l_quantity)
FROM customer, orders, lineitem
WHERE o_orderkey IN (
            SELECT l_orderkey
            FROM lineitem
            GROUP BY l_orderkey having
                  SUM(l_quantity) > :1 )
      AND c_custkey = o_custkey
      AND o_orderkey = l_orderkey
GROUP BY c_name, c_custkey, o_orderkey,
       o_orderdate, o_totalprice
ORDER BY o_totalprice desc, o_orderdate;
```

Listing 2: TPC-H Query 18

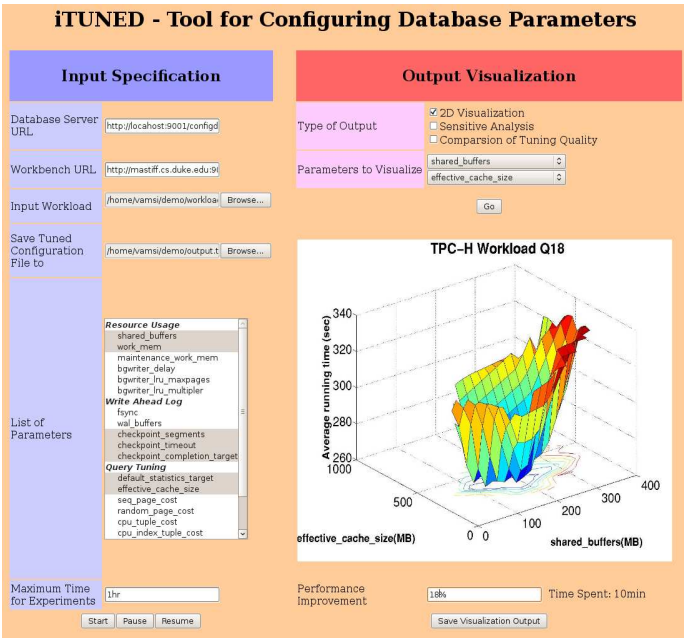

Figure 2: Partial 2D visualization of response surface for TPC-H Q18 (SF=1), 10 minutes after start (Scale Factor: 1)

3.1 Interacting with iTuned

iTuned has the ability to show progressive analysis as it conducts experiments during a tuning session. Suppose a DBA starts a tuning session by specifying the list of configuration parameters, SQL workload, database servers, and time budget through iTuned's front-end interface (see the left side of Figures 2-5).

In a hypothetical scenario, the DBA checks back after 5 minutes; and iTuned has nothing to report yet. When she checks back after 10 minutes, iTuned shows an intuitive visualization of the performance impact of each parameter along with a configuration setting that is 18% better than the current one (Figure 2). After 30 minutes, iTuned shows a 35% better configuration (Figure 3); and after the time budget of 1 hour is exhausted, iTuned reports a 52% better configuration (Figure 4). If the user is satisfied earlier than the specified time budget, then she can choose to terminate the tuning session (which will stop iTuned from running more experiments). Alternately, based on the initial results produced, the DBA may choose to alter the course of tuning by adding or dropping configuration parameters from the session.

3.2 Complexity of Parameter Tuning

We will demonstrate some real response surfaces that show the complexity and non-intuitive nature of performance tuning. The output in Figure 4 is a response surface that shows how the performance of the complex TPC-H Q18 depends on the shared_buffers and effective_cache_size parameters in PostgreSQL. The setting of shared_buffers is the size of PostgreSQL's main buffer pool for caching disk blocks. The value of effective_cache_size gives PostgreSQL an estimate of how much memory is available in the OS for caching disk blocks. The settings of both parameters are used by PostgreSQL's query optimizer while costing query execution plans. Some observations from Figure 4 are:

- The surface is complex and nonmonotonic. The query optimizer picks different plans in different regions. The plans picked at higher settings of shared_buffers and effective_cache_size turn out to be bad plans compared to lower settings.

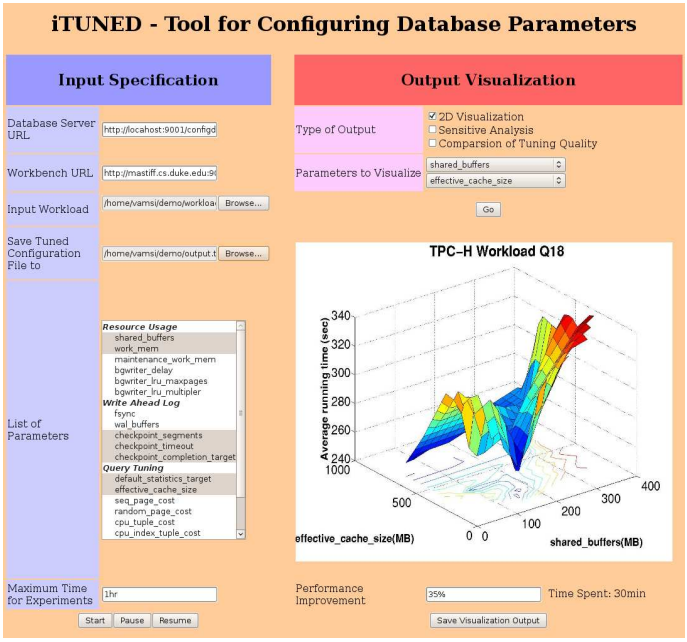

Figure 3: Partial 2D visualization of response surface for TPC-H Q18, 30 minutes after start (Scale Factor: 1)

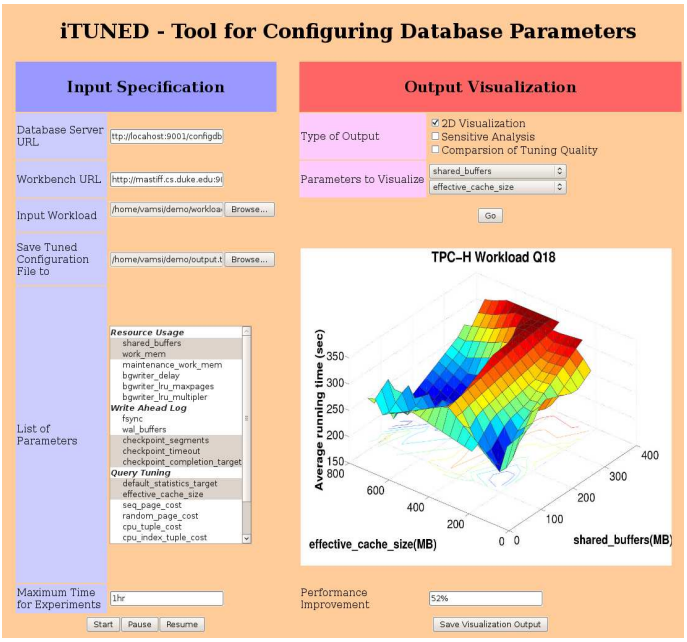

Figure 4: 2D visualization of response surface for TPC-H Q18, 1 hour after start (Scale Factor: 1)

- Performance drops sharply as shared_buffers is increased beyond 20% (200MB) of available memory; causing an "increase buffer pool size" rule of thumb to degrade performance.
- The effect of changing effective_cache_size is different for different settings of shared_buffers. Surprisingly, the best performance comes when both parameters are set low.

3.3 Sensitivity of Configuration Tuning

We will demonstrate how performance is sensitive to different parameter settings depending on the workload. For example, we found that performance is very sensitive to the shared_buffers and checkpoint-related parameters in transactional workloads, whereas work_mem and effective_cache_size are high-impact parameters in data warehousing workloads. Using iTuned's sensitivity analysis plots (like Figure 5), the DBA can identify such effects and focus on tuning the important parameters for the current database workload.

The last part of our demonstration will illustrate iTuned's superior performance compared to existing techniques. We will compare iTuned's tuning quality and time with default settings in the database system, rule-based settings from experts [6], and (approximate) optimal settings found through brute force. We will also demonstrate the scalability of iTuned to large numbers of parameters and complex SQL workloads.

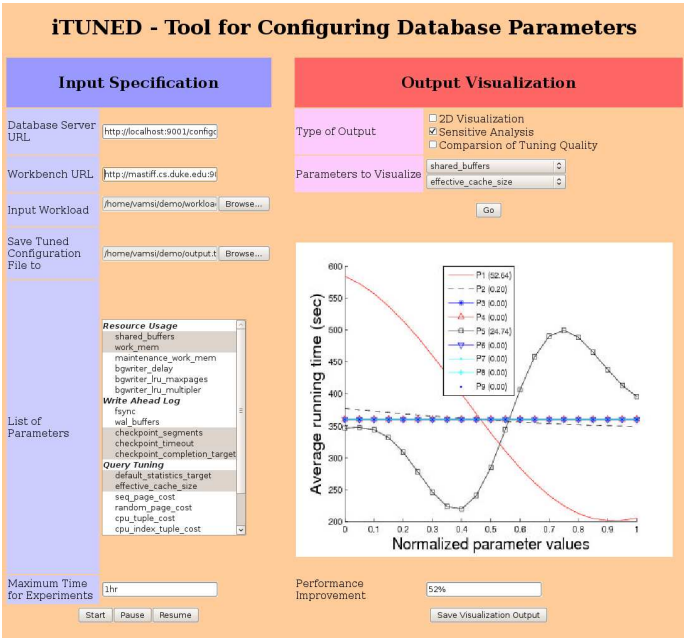

Figure 5: Sensitivity-analysis plot for TPC-H Q18, P1: shared_buffers, P3: _cache_size (Scale Factor: 10)

4. REFERENCES

[1] S. Babu, N. Borisov, S. Duan, H. Herodotou, and V. Thummala. Automated Experiment-Driven Management of (Database) Systems. In *Proceedings of the XII Workshop on Hot Topics in Operating Systems (HotOS)*, May 2009.

[2] S. Duan, V. Thummala, and S. Babu. Tuning Database Configuration Parameters with iTuned. In *Proceedings of Very Large Data Bases (VLDB)*, Aug 2009.

[3] R. B. Gramacy. tgp: An R package for Bayesian Nonstationary, Semiparametric Nonlinear Regression and Design by Treed Gaussian Process Models. In *Journal of Statistical Software*, 2007.

[4] IBM DB2 Configuration Advisor, 2004. `http://www.ibm.com/developerworks/data/library/techarticle/dm-0605shastry/index.html`.

[5] PostgreSQL Tuning Wizard, 2008. `http://pgfoundry.org/projects/pgtune`.

[6] Tuning Your PostgreSQL Server, 2009. `http://wiki.postgresql.org/wiki/Tuning_Your_PostgreSQL_Server`.

[7] Solaris Containers, 2009. `http://www.sun.com/bigadmin/content/zones/index.js`.