Characterizing Resource Sensitivity of Database Workloads

Rathijit Sen　　Karthik Ramachandra

Gray Systems Lab
Microsoft Corporation
rathijit.sen@microsoft.com, karam@microsoft.com

ABSTRACT

The performance of real world database workloads is heavily influenced by the resources available to run the workload. Therefore, understanding the performance impact of changes in resource allocations on a workload is key to achieving predictable performance. In this work, we perform an in-depth study of the sensitivity of several database workloads, running on Microsoft SQL Server on Linux, to resources such as cores, caches, main memory, and non-volatile storage. We consider transactional, analytical, and hybrid workloads that model real-world systems, and use recommended configurations such as storage layouts and index organizations at different scale factors.

Our study lays out the wide spectrum of resource sensitivities, and leads to several findings and insights that are highly valuable to computer architects, cloud DBaaS (Database-as-a-Service) providers, database researchers and practitioners. For instance, our results indicate that throughput improves more with more cores than with more cache beyond a critical cache size; depending upon the compute vs. I/O activity of a workload, hyper-threading may be detrimental in some cases. We discuss our extensive experimental results and present insights based on a comprehensive analysis of query plans and various query execution statistics.

1. INTRODUCTION

Real-world database workloads exhibit a wide spectrum of resource utilization characteristics, depending upon several factors such as the hardware, nature of the workload, database configuration, data sizes and so on. Furthermore, different workloads exhibit different sensitivity characteristics with respect to the available hardware resources. For instance, a certain workload might tolerate lower cache sizes without much impact on throughput whereas it might quickly degrade if given fewer cores. A different workload might withstand fewer cores but might be highly sensitive to write I/O bandwidth.

Characterizing the resource sensitivity of database workloads is important for several reasons. It is valuable to computer architects for designing specialized database servers—this is attractive to DBaaS (Database-as-a-Service) providers due to economies of scale in datacenters hosting such services. It is also valuable to database researchers and users for improving query processing algorithms to make better use of available resources, or adapt gracefully to changes in resource availability. Further, it also aids users in tuning and configuring database system resources for their workloads.

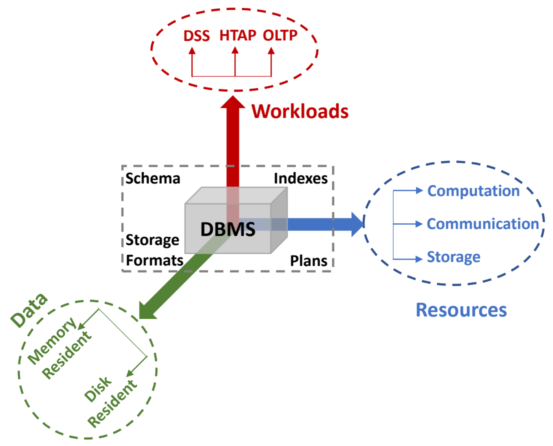


Figure 1: Dimensions for DBMS performance analysis.

Recent trends clearly indicate the maturity and increasing adoption rates of DBaaS offerings. At cloud scale, database servers in datacenters need to be able to efficiently handle diverse database workloads, in contrast to on-premise servers that handle specific workloads. Further, the nature of workloads themselves are undergoing many changes. With the advent of BigData, Operational Analytics, and the ever-increasing rate of online transactions, the scale and complexity of database workloads is higher than ever before. These factors necessitate a comprehensive study that covers a broad range of diverse workloads so as to identify key insights and areas where inefficiencies can be addressed.

To this end, we have conducted an in-depth study that characterizes resource utilization and resource sensitivity of database workloads. Figure 1 gives an idea of the different aspects that need to be considered and evaluated in order to achieve a thorough, useful characterization. Our study covers a broad space with multiple dimensions, depicted in Figure 1. In particular, our study includes (i) different kinds of workloads—transactional (OLTP), analytical (DSS), and hybrid (HTAP), (ii) different resources—cores, caches, memory and disk, (iii) different data sizes ranging from those that fit in memory to those that do not, and (iv) different database configurations—row or column storage formats, index structures etc. We also analyze query plans and their impact on performance evaluation.

Thanks to recent advancements, modern hardware exposes several knobs that were hitherto unavailable, making it now possible to conduct such studies. Our study, conducted on Microsoft SQL Server on Linux, makes use of Intel's Cache Allocation Technology (CAT) for measuring last-level cache space utilization, on-chip performance counters for measuring DRAM bandwidth utilization, Linux device statistics and cgroup resource limits for measuring non-volatile storage (SSD) bandwidth utilization, and restricted core affinities for measuring core utilization.

The results of our study bring out the wide variety in resource sensitivities of database workloads with respect to different resources based on the nature of the workload and their sizes in relation to the main memory capacity of the database server. For example, we find that hyper-threading may degrade the performance of compute-intensive analytical workloads with data that fits in memory, but may improve the performance of similar workloads with larger data sizes (that do not fit in memory). We also find that a larger-sized database may achieve higher transactions-per-second (TPS) than a smaller-sized database on a transactional workload due to reduced contention for shared data.

Although there have been earlier studies on resource utilization of database workloads [1–9], they have either mostly focused on a single kind of workload (e.g., OLTP, OLAP), or a single database configuration or hardware configuration. While such studies are useful to gain a deeper understanding of the specific configuration or workload, they do not provide a holistic view of the behavior of the database system. Furthermore, the study of sensitivity of different workloads to specific resources has also not received much attention in prior work. Through our study, we establish that studying a single class of database workloads or a single size of databases can sometimes be misleading because it does not reveal the full spectrum of resource utilizations and sensitivities. This is particularly important in the context of cloud servers, as they have to be designed to host diverse user workloads.

The main contributions of this paper are:

1. We present the results of our comprehensive analysis of sensitivities to resources such as cores, cache capacity, degree of parallelism, and storage bandwidths for transactional, analytical, and hybrid database workloads at different scale factors. In general, we find that the number of cores, memory capacity, and non-volatile storage bandwidths are critical system resources affecting database performance; cache capacity and memory bandwidths are usually under-utilized.
2. To the best of our knowledge, this is the first work to use Intel's Cache Allocation Technology (CAT) to characterize the last-level cache utilization of a commercial database management system running a wide spectrum of workloads. Our results suggest that small caches may be sufficient for reasonable performance and that analytical workloads may benefit more from increased cache allocation than transactional ones.
3. We discuss the importance of database-specific features and configurations such as storage layouts and index organizations for different types of workloads and their impact on resource utilization. These configurations can be useful to performance analysts as they evaluate alternative algorithms and hardware configurations for improved performance.
4. We show that, since advanced database management systems often adapt internally in response to changed resource allocations, e.g., generating different query plans based on available parallelism, a coordinated hardware-software optimization strategy is needed to better utilize available system resources.
5. We identify and enumerate the common pitfalls in conducting such studies on database workloads, and the necessary steps to be taken to ensure reliable and meaningful results.

The rest of this paper is organized as follows. Section 2 presents background information on our workloads and discusses database-specific features that affect performance and resource utilizations. Section 3 describes our experimental setup. Sections 4, 5 and 6 analyze sensitivities to cores, last-level cache (LLC) size, and bandwidth (DRAM and SSD) for our workloads. Sections 7 and 8 analyze query performance sensitivity to degree of parallelism and memory capacity. Section 9 lists some pitfalls to be avoided for database performance analysis studies. Section 10 suggests potential research questions. Section 11 briefly discusses related work and Section 12 concludes the paper.

2. DATABASE WORKLOADS

Real world relational database systems and applications are quite complex, and can be set up and configured in many different ways. Based on the nature of use, a database systems expert makes several design choices. The choice of storage layout, design of the schema and tables, usage of index-structures and materialized views are some of the important choices that significantly impact performance. In this work, we present a characterization of database workloads that resemble real-world workloads to a large extent. Therefore, we consider commonly used design principles and configurations based on established database systems benchmarks. In this section, we will describe the different benchmark workloads and database configurations that we have used in our study.

Database workloads can be broadly classified into OLTP (On-Line Transaction Processing) and OLAP (On-Line Analytical Processing) workloads. A third category, referred to as HTAP (Hybrid Transactional and Analytical Processing) has recently gained a lot of interest in the database community [10–13]. We have considered all the three types of workloads in our study. We now describe each type of workload and the corresponding database configuration we have used. These are based on recommended design guidelines used in real applications.

2.1 OLTP Workloads

Operational, or online transaction processing (OLTP) workloads are characterized by small, interactive transactions that generally require sub-second response times. OLTP systems, often referred to as transactional systems, are designed to process small, interactive queries and data modifications with high concurrency requirements. Modifications predominantly affect single records, and most queries are limited to simple joins.

We have used two OLTP benchmarks in our study:

- **TPC-E:** The TPC Benchmark E [14] is a mixture of read-only and update intensive transactions that simulate the activities found in complex OLTP applications. The benchmark simulates the transactional workload of a brokerage firm with customers who generate transactions related to trades, account inquiries, and market research. The brokerage firm in turn interacts with financial markets to execute orders on behalf of customers and updates relevant account information. Different transaction types are defined to simulate interactions of the firm with its customers and business partners, and have varying runtime requirements.

  The focus of the benchmark is the central database that executes transactions related to the firm's customer accounts. Although the underlying business model of TPC-E is a brokerage firm, the database schema, data population, transactions, and implementation rules are broadly representative of modern OLTP systems.

- **Azure SQL Database Benchmark:** The Azure SQL Database Benchmark (ASDB) [15] is a synthetic workload and database schema designed to measure the performance of database operations which occur frequently in OLTP systems. The benchmark runs against a database comprised of tables that fall into three categories: fixed-size, scaling, and growing. Fixed-size tables have a constant number of rows. Scaling tables have a cardinality that is proportional to database size, but does not change during the benchmark. The growing table is sized like a scaling table on initial load, but then the cardinality changes in the course of running the benchmark as rows are inserted and deleted.

  This benchmark is designed primarily for cloud database services such as Azure SQL Database [16]. Azure SQL Database is a fully managed relational database service, and it offers multiple service tiers such as Basic, Standard and Premium. These tiers are primarily differentiated by a range of performance level and storage size choices, and price. All service tiers provide flexibility in changing performance level and storage size [17].

The key metric that is relevant for OLTP workloads is Transactions-Per-Second (TPS). We measure TPS for both TPC-E and ASDB in this paper.

2.1.1 *Database design and configuration*

As mentioned earlier, OLTP workloads are characterized by lots of interactive database operations (queries, updates, inserts, deletes) with high concurrency requirements. For such workloads, it is generally expected that atomicity, consistency, isolation, and durability (ACID) properties will be maintained while providing sub-second response times. Database design for these business-critical systems are driven by the above constraints, and it becomes more challenging with large volumes of data, users, and transactions rates. We now describe how OLTP databases are typically designed and configured to support such workloads.

- Schema design: OLTP databases generally store data in a normalized form which reduces data redundancy and aids consistency. Database designs generally start with third normal form (3NF) enforced with referential integrity (RI) constraints. This level of normalization may lead to queries with more joins, so they selectively deviate to second normal form (2NF) when necessary to enhance performance [18].
- Storage layout: OLTP systems store data in a row-oriented form since queries generally affect single (or a few) rows, and access most attributes.
- Index structures and constraints: B-Tree indexes are often used in OLTP databases both to avoid table scans as well as to increase concurrency. However, using too many indexes has the downside of reducing the performance of DML (updates/inserts/deletes) statements due to index maintenance overhead. Uniqueness, not-null, and check constraints are often used in OLTP systems to maintain data consistency.

2.2 Decision Support Workloads

On-Line Analytical Processing (OLAP) and Data Warehousing (DW) are decision support technologies. Their goal is to enable enterprises to gain competitive advantage by exploiting the ever-growing amount of data that is collected and stored in corporate databases and files for better and faster decision making [19]. Symmetric multi-processing (SMP) data warehousing [20] is a common architecture for data warehouses up to a few terabytes in size. These systems are characterized by a single instance of a RDBMS sharing all resources.

In this paper, we use the SMP DW configuration and the TPC Benchmark H (TPC-H), a standard decision support benchmark [21]. It consists of a suite of business oriented ad-hoc queries and few data modifications. This benchmark models decision support systems that examine large volumes of data, execute queries with a high degree of complexity, and give answers to critical business questions. The common metric used for analytical workloads is Queries-Per-Second (QPS) or Queries-Per-Hour (QPH).

2.2.1 *Database design and configuration*

OLAP systems are characterized by queries that scan larger numbers of rows, large ranges of data, and may perform aggregations or return relatively large results for the purposes of analysis and reporting. Data warehouses are also characterized by relatively large data loads versus small transaction inserts, updates, or deletes. Driven by the above requirements, the design of such databases typically follow the below guidelines.

- Schema design: Database schemas are often designed to be less normalized. This is achieved by pre-materializing

Table 1: Database designs for different workload types.

Workload Type	Schema	Storage layout	Auxiliary structures
OLTP	Normalized (3NF/2NF)	Row store	B-Tree indexes, constraints
Decision support	Denormalized	Column store	Column-store indexes, compression, materialized views
HTAP	Normalized (3NF/2NF)	Row store	Updateable non-clustered column-store indexes

frequent joins and aggregations at the cost of data redundancy. Since DML statements are rare in such workloads, data redundancy can be tolerated. Typically, decision support databases use a star-schema or a snowflake-schema with some large fact tables and other dimension tables.

- Storage layout: OLAP systems typically store data in a columnar form. For analytical workloads, columnar storage layout provides several benefits over row-stores [22]. Column stores can achieve high compression rates, which minimizes the I/O bottleneck and improves query performance due to a smaller memory footprint. They can use batched execution of query operators and make use of architectural features such as SIMD instructions to improve performance. Analytical queries often project a few columns from a table, which reduces the I/O due to columnar storage formats.
- Index structures and constraints: OLAP implementations sometime use columnar storage for fact tables and row store for dimension tables (we use fully columnar formats in our experiments). The row store tables use B-tree indexes while fact tables use columnar index structures. Also, DW systems use indexed views (also known as materialized views) which significantly improve query performance but increase database size and redundancy. Constraints are avoided as far as possible.

2.3 HTAP Workloads

Hybrid Transaction/Analytical Processing (HTAP) is an emerging application architecture that aims to "break the wall" between transaction processing and analytics. It enables more informed and "in-business real-time" decision making. HTAP is also referred to as Real-time operational analytics. Traditionally, businesses have had separate systems for operational (i.e. OLTP) and analytics workloads. For such systems, Extract, Transform, and Load (ETL) jobs regularly move the data from the operational store to an analytics store. The analytics data is usually stored in a data warehouse dedicated to running analytics queries. While this solution has been the standard, it is (a) more complex (due to complex ETL processes), (b) more expensive (due to the need for additional hardware and software), and (c) is not real-time (i.e. analytics runs on stale data).

Real-time operational analytics offers a solution to these challenges by enabling analytical queries on the transactional database. There is no staleness when analytics and OLTP workloads run on the same underlying table. For scenarios that can use real-time analytics, the costs and complexity are greatly reduced by eliminating the need for ETL and the need to purchase and maintain a separate data warehouse.

In order to characterize a real-world HTAP workload, we have adapted the TPC-E benchmark by augmenting a set of analytical queries on the TPC-E schema. These queries operate on the large, fast-growing tables in the schema with large scans, joins and aggregations. We concurrently run the transactional workload and these analytical queries, and measure the transaction throughput as well as queries-per-hour (QPH).

2.3.1 Database design and configuration

One of the ways to design a database for HTAP workloads is to use an updateable columnstore index on a rowstore table [23]. The column store index maintains a copy of the data, so the OLTP and analytics workloads run against separate copies of the data. This minimizes the performance impact of both workloads running at the same time. The database maintains index changes so OLTP changes are always up-to-date for analytics. With this design, it is possible and practical to run analytics in real-time on up-to-date data.

Table 1 summarizes database designs for different workloads that are typically used. We have followed these designs in our experiments, and as we will show, these choices play an important role in the performance of different database workloads.

3. EXPERIMENTAL SETUP

Our test system is a dual-socket Lenovo Thinkstation P710 with Xeon E5-2620 v4 (Broadwell) processors and 64 GB DDR4 memory. Each socket has 8 physical cores and 20 MB LLC. We enable hyper-threading for all our experiments, resulting in a total of 32 logical cores for the server. Each socket has a theoretical peak memory bandwidth of 68.3 GB/sec. However, due to only one-third of the memory channels being populated, we expect the maximum achievable bandwidth to be less than that. The peak QPI data transfer rate is 8 GT/sec (that is, 32 GB/sec). We run with turbo boost enabled (peak frequency: 3.0 GHz, nominal: 2.1 GHz). We use a 1.2 TB Intel 750-Series NVMe SSD, that supports up to 2500 MB/sec sequential read bandwidth and up to 1200 MB/sec sequential write bandwidth, for hosting the database and log files. The OS and swap area are installed on a separate 512 GB NVMe device.

We run SQL Server on Linux [24] (version: 2017 RC1, Ubuntu 16.01.1 LTS) with analytical (TPC-H), transactional (ASDB and TPC-E), and hybrid (HTAP) workloads. Section 2 describes these workloads in more detail. Each database has two (four for TPC-H) scale factors, one of which fits into the available memory while the other does not. Table 2 shows the scale factors and sizes for all our workloads.

For TPC-H, we execute three concurrent query streams, except for experiments in Sections 7 and 8 where only one query stream executes. Each query stream runs all 22 TPC-H queries in a random order. We run other workloads for one hour for each experiment. ASDB runs with 128 client threads. TPC-E runs with 100 users. The HTAP workload

Table 2: Database scale factors and initial size. Databases not fitting within the 64 GB server memory are shaded.

Database	Scale Factor	Data (GB)	Index (GB)
ASDB	2000	51.13	0.21
	6000	153.36	0.64
TPC-E	5000	31.99	8.15
	15000	96.45	24.61
HTAP	5000	31.99	10.44
	15000	96.45	31.74
TPC-H	10	5.54	0.13
	30	12.93	0.23
	100	41.95	0.75
	300	127.94	2.25

also has 100 users, with 99 of them executing the transactional component and the remaining user executing the analytical component. The analytical component of HTAP repeatedly runs four distinct queries sequentially till the end of the hour.

We use the `iostat` utility to obtain SSD read and write bandwidths, and the Processor Counter Monitor utility [25] to obtain DRAM read and write bandwidths, counts of LLC misses, and instructions retired. All measurements are for average values taken over 1-second intervals.

4. SENSITIVITY TO NUMBER OF CORES

Broadly speaking, the nature of the workload largely influences the manner in which available cores are used by the database system. Analytical workloads and long running operations over larger data often use cores to achieve higher degrees of intra-query parallelism. In contrast, transactional workloads make use of cores to support more concurrent queries (inter-query parallelism), while scarcely using intra-query parallelism.

To study how performance is impacted as the number of allocated cores change, we restrict core affinities of all SQL Server processes and threads using the `cpuset` cgroup feature in Linux. Additionally, we restrict the maximum degree of parallelism[1] using SQL Server's resource governor settings. As we increase the number of allocated cores from 1 to 16, we first allocate cores on socket 0, with one logical core corresponding to each physical core, before allocating cores from socket 1. Finally, for 32 cores, we allocate the second logical core for all 16 physical cores.

Figure 2 (a, d, g, j) shows the average performance (QPS) for different workloads and scale factors as the number of allocated cores change. For these experiments, we run 3 concurrent query streams for TPC-H. Figure 2a shows that performance scales well with the number of cores for all scale factors. However, allocating the hyper-threaded second logical core for each physical core is detrimental at small scale factors, but beneficial at larger scale factors due to change in the amount of I/O operations involved. Relative to that with 32 cores, performance with 16 cores is at 1.72, 1.27, 0.93, 0.82, for scale factors 10, 30, 100, and 300 respectively.

For transactional workloads, ASDB and TPC-E, performance (TPS) scales well with the number of physical cores. With the additional logical cores, performance also improves (5–6.8% for ASDB, 16.7–24.2% for TPC-E). ASDB shows similar behavior for both scale factors whereas TPC-E shows better performance at the larger scale factor (15000) despite incurring more I/O operations. Table 3 shows the relative total wait times for different lock and latch operations for TPC-E with SF=15000 compared to that with SF=5000. As expected, there are significantly more PAGEIOLATCH waits for SF=15000 as the data needs to be fetched from secondary storage (SSD). However, once the data is in memory, the total LOCK/LATCH/PAGELATCH waits are almost half at SF=15000 compared to those at SF=5000. The breakdown shows that the LATCH waits do increase, but are more than compensated by reductions in waits for LOCKs and PAGELATCHes. The takeaway is that for transactional workloads, performance may change in unexpected ways at different scale factors due to changes in contention patterns to shared data, particularly in systems having fast, high-bandwidth secondary storage. We do not observe this effect for read-only analytical workloads.

[1]The max. number of threads to use in a query [26].

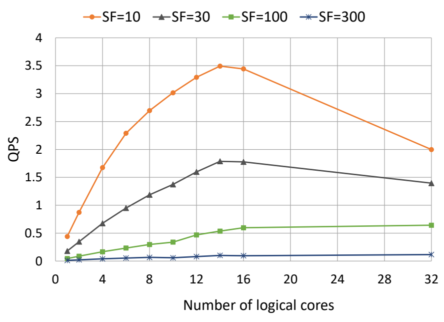
Table 3: Lock and Latch wait times for TPC-E at SF=15000 relative to that at SF=5000.

Wait Type	Description	Ratio
LOCK	Wait for Shared/Update/Exclusive/Intent Shared/Intent Update/Intent Exclusive Lock	0.15
LATCH	Wait for Shared/Update/Exclusive Latch	2.06
PAGELATCH	Wait for Shared/Update/Exclusive access to latch on buffer that is in a non-I/O request	0.56
Σ	Wait for LOCK/LATCH/ PAGELATCH	0.49
PAGEIOLATCH	Wait for Shared/Update/Exclusive access to latch on buffer that is in an I/O request	74.61

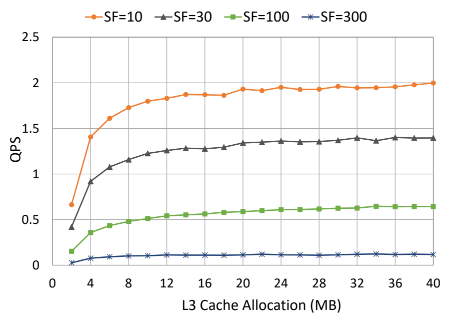
Likewise, the analytical (DSS) and transactional (OLTP) components of HTAP behave differently at different scale factors. For SF=15000, DSS performs less and OLTP performs better than for SF=5000. But all components benefit from increased core allocations and, like TPC-E (and ASDB), using the additional hyper-threaded logical cores is beneficial for this workload.

5. SENSITIVITY TO LLC CAPACITY

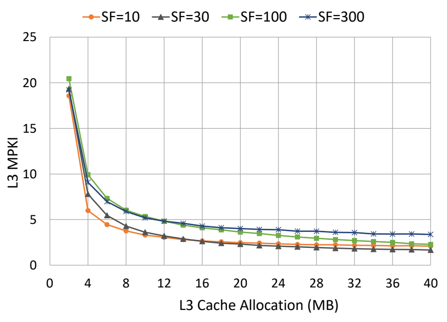
Our processors support Intel's Cache Allocation Technology [27]. This allows system software to designate LLC space where cores can allocate data into and evict data from. This involves a two-step mapping: cores are assigned a Class-Of-Service (COS) and LLC space is assigned to the COS. We use the freely-available `pqos` utility [28] to change LLC allocations. We keep the core-to-COS mapping unchanged (all cores have the same COS), but change the LLC allocation for that COS. The allocated LLC space is divided equally between the sockets. On our system, with a 20 MB LLC per socket, the bitmask length is 20 bits corresponding to an allocation granularity of 2 MB (1 MB per socket).



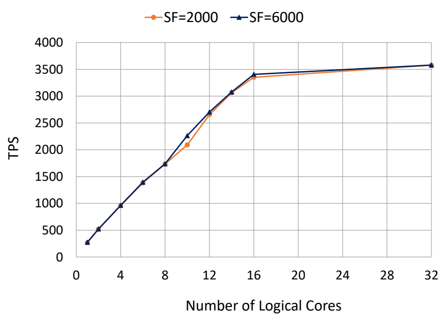
(a) TPC-H: Core Sensitivity



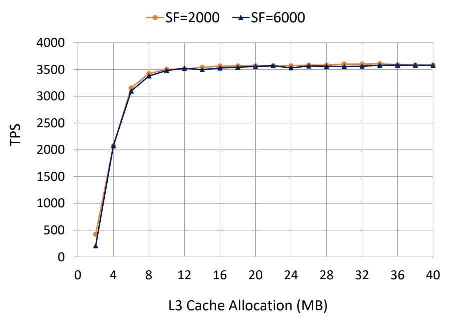
(b) TPC-H: LLC Sensitivity



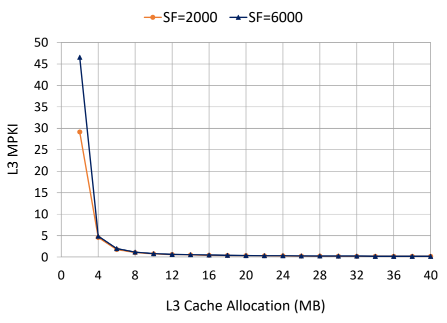
(c) TPC-H: LLC MPKI



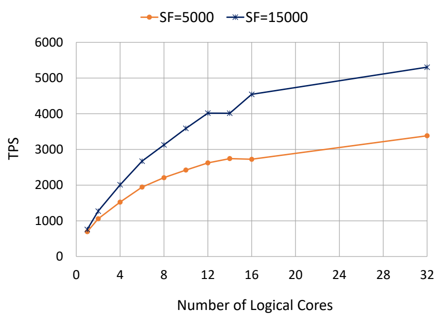
(d) ASDB: Core Sensitivity



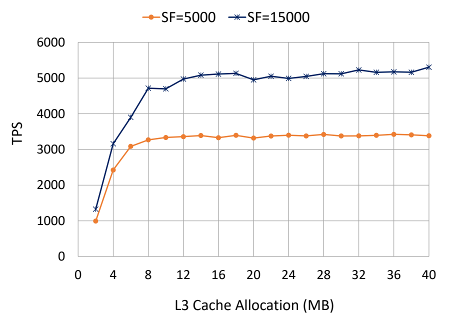
(e) ASDB: LLC Sensitivity



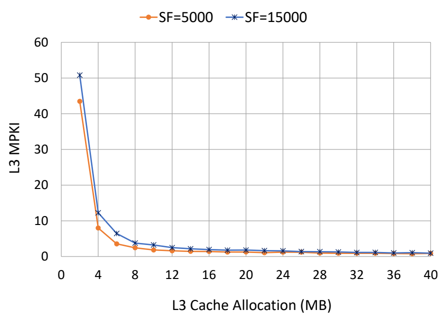
(f) ASDB: LLC MPKI



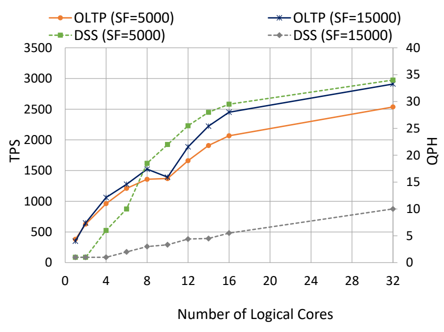
(g) TPC-E: Core Sensitivity



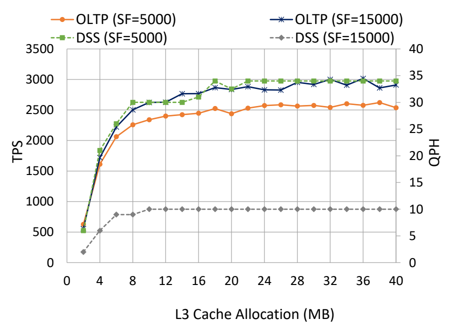
(h) TPC-E: LLC Sensitivity



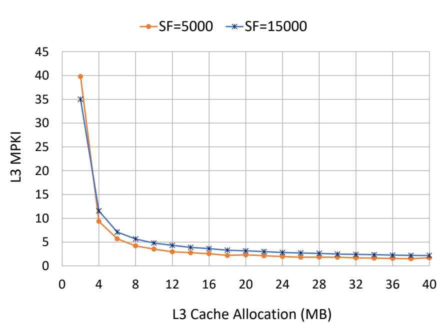
(i) TPC-E: LLC MPKI



(j) HTAP: Core Sensitivity



(k) HTAP: LLC Sensitivity



(l) HTAP: LLC MPKI

Figure 2: Core and Cache Sensitivities for our workloads. (a), (d), (g), (j): Average workload performance vs number of logical cores (with 40 MB LLC allocated); (b), (e), (h), (k): Average workload performance vs size of LLC allocation (with 32 cores allocated); (c), (f), (i), (l): Average cache performance vs size of LLC allocation (with 32 cores allocated). Increasing core allocations to more than 8 crosses the socket boundary.

As we increase LLC allocations, we allocate ways such that they are a superset of the previous allocation. So, for example, we use the bitmask 1 for 2 MB (total LLC across sockets), 3 for 4 MB, 7 for 6 MB, and so on.

While CAT restricts allocation and eviction spaces, accesses to data outside the assigned space will still be cache hits. To reduce its effect on our observations, we load the database after changing the allocation, conduct experiments with increasing allocations, and reboot the system before starting experiments with the smallest allocation (2MB).

Figure 2 (b, c, e, f, h, i, k, l) shows how average throughputs and cache miss rates (Misses Per Kilo Instruction, or, MPKI) change with LLC allocation. Each data point corresponds to a particular amount of LLC, divided equally between the two sockets, that is allocated to all cores in the system. While workload performance generally increases with LLC allocation, the increase at smaller sizes is more dramatic than at larger sizes. For example, for TPC-H SF=100, increasing LLC allocation from 2 MB to 10 MB results in a speedup of 3.4, but a further increase to 40 MB improves performance by a (relatively) modest 26%. Performance increase is due to a decrease in the cache MPKI as allocations increase, with a more significant change in MPKI at smaller allocations than at larger ones.

Table 4: Sufficient LLC capacity with 32 cores.

Workload	**SF**	**Perf**$\geq$90%	**Perf**$\geq$95%
ASDB	2000	8 MB	8 MB
	6000	8 MB	10 MB
TPC-E	5000	6 MB	8 MB
	15000	12 MB	14 MB
HTAP	5000	16 MB	18 MB
	15000	10 MB	14 MB
TPC-H	10	10 MB	14 MB
	30	10 MB	16 MB
	100	16 MB	22 MB
	300	12 MB	12 MB

The miss rate curves exhibit knees [29] at small cache sizes for our workloads. The 99^{th}-percentile latency for ASDB (not shown) also exhibits a similar knee. Table 4 shows the LLC sizes needed for the performance to be at least 90% and at least 95% respectively of that with the full 40 MB LLC allocation. We agree with prior studies [30–32] that current servers have over-provisioned LLCs in terms of cache-to-core ratio, but also find that analytical and hybrid workloads need a somewhat larger cache than transactional ones.

6. STORAGE BANDWIDTH SENSITIVITY

Figure 3 shows the average SSD and memory bandwidths used by two of our workloads, one analytical (TPC-H) and the other transactional (ASDB). SSD bandwidth utilizations increase with performance in both cases. Conversely, a low bandwidth limit will severely limit performance despite using a large number of cores. DRAM bandwidth utilization also increases with performance when it is due to a larger number of cores. When performance increases due to a larger cache size, DRAM bandwidth utilization drops due to the reduced number of cache misses. We show these two trends separately in Figure 3. Thus, increasing cores and decreasing caches [31] will each result in increasing the DRAM bandwidth requirement, but this appears to be feasible as currently the available bandwidth is under-utilized.

Figure 4 shows the cumulative distributions of SSD and DRAM bandwidths for our workloads with full core and LLC allocations. TPC-H (SF=300) shows the largest bandwidth utilization for both DRAM and SSD, followed by HTAP (SF=15000). Thus, one needs to run analytical queries on large data sets to stress the storage bandwidths. The bandwidth requirements for the transactional components are lower, but a significant portion of their SSD bandwidth use is for writes whereas it is mostly reads for analytical components.

To study sensitivity to SSD bandwidth allocations, we use `systemctl`'s `BlockIOReadBandwidth` and `BlockIOWrite-Bandwidth` properties, that in turn uses Linux's `cgroup` Block IO Controller [33] capabilities, to impose bandwidth limits. Figure 5 shows that TPC-H (SF=300) performance, with full core and LLC allocations, is significantly affected by the read bandwidth limit. But the QPS response curve is non-linear with diminishing returns for increased bandwidth allocations. This makes it non-trivial to optimally select the proper allocation (SLO level, and consequently, the price point in a cloud environment) for a required performance target. For example, a simple linear model would suggest allocating around 1000 MB/sec for a QPS of $\sim$0.08 whereas the same performance could be reached with around a 800 MB/sec allocation—a 20% reduction.

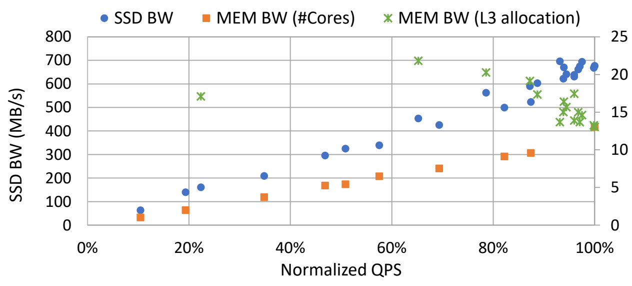
We also studied the impact of write bandwidth limits on transactional workloads. As an example, the TPS for ASDB (SF=2000) drops by 6% and 44% when the write bandwidth is limited to 100 MB/sec and 50 MB/sec respectively, even though it mostly fits in memory. Transactional workloads experience significant (blocking) logging activity and data updates that contribute to their sensitivity to write bandwidth.

7. SENSITIVITY TO INTRA-QUERY PARALLELISM

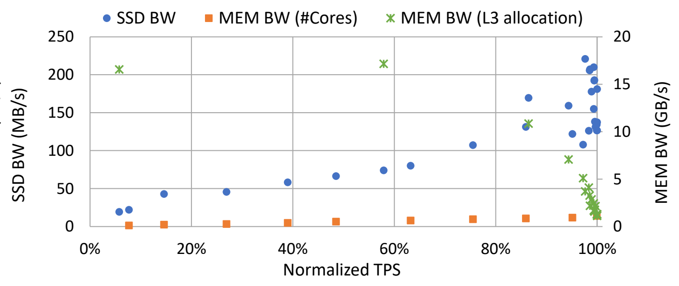
A single query can be evaluated by decomposing its operations into tasks that execute concurrently on multiple processors. Such an evaluation strategy is referred to as intra-query parallelism. SQL Server automatically detects the best degree of parallelism for a query based on criteria such as the availability of multiple processors and threads, the type of the query or data definition language (DDL) operation, and the data size.

The query optimizer decides the degree of parallelism for a given query in a cost-based manner. Long running operations such as analytical queries or index maintenance operations that use CPU cycles heavily are typically benefited by a parallel query plan. Joins of large tables, large aggregations and sorting are good candidates. For simple queries that are often found in OLTP applications, the additional overheads of coordination required for parallel plans outweigh the potential performance boost.

Therefore, for our study of parallelism, we only consider analytical workloads (TPC-H). We use the MAXDOP (Max. Degree Of Parallelism) query hint [26] to limit the parallelism as necessary. Figure 6(a, b, c, d) shows, for 4 different

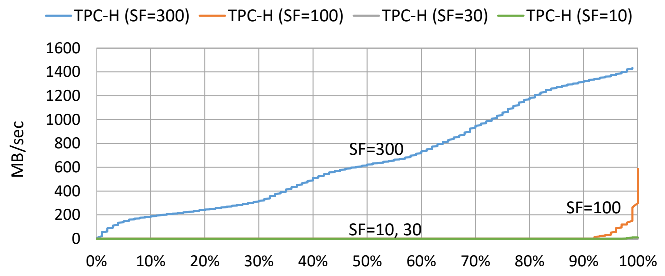


(a) TPC-H (SF=300)

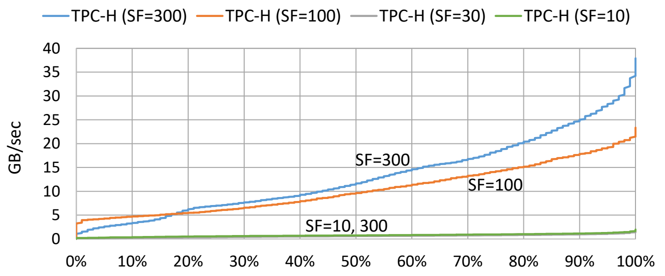


(b) ASDB (SF=6000)

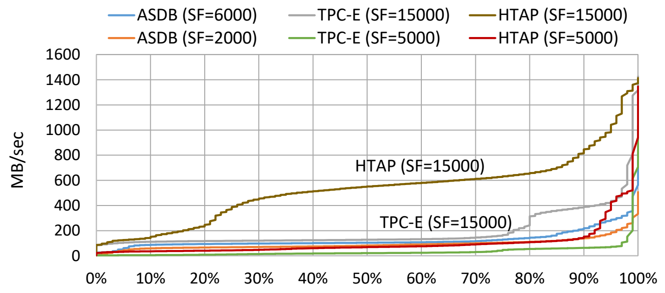
Figure 3: Average bandwidth utilizations for TPC-H and ASDB.



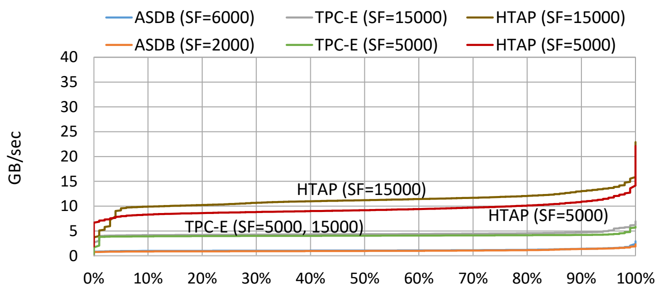
(a) SSD bandwidth: TPC-H



(b) DRAM bandwidth: TPC-H



(c) SSD bandwidth: ASDB, TPC-E, HTAP



(d) DRAM bandwidth: ASDB, TPC-E, HTAP

Figure 4: Cumulative distributions of SSD and DRAM bandwidth use with full core and LLC allocations.

scale factors, the relative speedup for each of the 22 TPC-H queries with different MAXDOP settings using MAXDOP=32 as the baseline. We also limit the number of cores to the same number as MAXDOP. Only one stream of queries execute, in a random order, for these experiments. We observe that in general, performance sensitivity to parallelism varies widely depending on both the query and scale factor.

For scale factor 10, some queries (such as queries 2, 6, 14, 15, and 20) are completely insensitive to parallelism, while the other queries are more sensitive. It turns out that for some queries in smaller scale factors, the query optimizer may always choose a serial plan, because it deduces that the overhead of parallelism outweighs the benefits. For larger scale factors such as 100 and 300, we observe that almost all queries show a clear improvement between a serial plan (maxdop 1) and others, implying that it is almost never a good idea to run these queries in a serial manner.

Listing 1: TPC-H Query 20 (Part Promotion)

```
SELECT S_NAME, S_ADDRESS
FROM   SUPPLIER, NATION
WHERE  S_SUPPKEY IN
       ( SELECT PS_SUPPKEY
         FROM   PARTSUPP
         WHERE  PS_PARTKEY IN
                ( SELECT P_PARTKEY
                  FROM   PART
                  WHERE  P_NAME like 'lemon%%'
                ) AND PS_AVAILQTY >
                ( SELECT 0.5 * sum(L_QUANTITY)
                  FROM   LINEITEM
                  WHERE  L_PARTKEY = PS_PARTKEY
                  AND    L_SUPPKEY = PS_SUPPKEY
                  AND    L_SHIPDATE >= '1993-01-01'
                  AND
L_SHIPDATE<dateadd(yy,1,cast('1993-01-01' as date)))
AND S_NATIONKEY = N_NATIONKEY
AND N_NAME = 'ALGERIA'
ORDER BY S_NAME
```

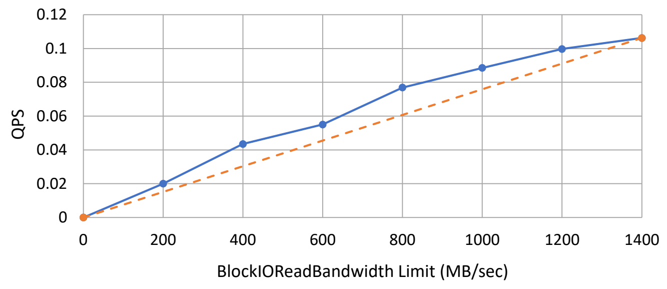


Figure 5: Non-linear QPS response for TPC-H (SF=300) with different limits for read I/O bandwidth of non-volatile storage (SSD). The dashed line shows a hypothetical linear response curve for reference.

The important observation here is that the database system is adapting to the available degree of parallelism by choosing alternate execution strategies for the same query. As a concrete example, we consider query 20 (shown in Listing 1), which is completely insensitive to parallelism in lower scale factors (10 and 30), but shows up to 10x speedup between MAXDOP=1 and MAXDOP=32 at scale factor 300. We analyzed the query plans for query 20 at different scale factors and MAXDOP settings. The serial query plan (MAXDOP=1) is shown in Figure 7a. This plan is chosen for all MAXDOP settings at scale factor 10 and 30, and also for the MAXDOP=1 setting at scale factor 300. The query plan with MAXDOP=32 is shown in Figure 7b.

It can be seen that the shape of the MAXDOP=32 plan varies quite drastically compared to the MAXDOP=1 plan. In particular, the differences can be summarized as follows.

1. The MAXDOP=32 plan (Figure 7b) uses parallel implementations for all operators (indicated by the double arrow symbol) which significantly improves performance compared to the MAXDOP=1 plan (Figure 7a) at higher scale factors.
2. The number of join operations and the join orders are different in the plans. Further, the join algorithms chosen are also different—the MAXDOP=32 plan uses parallel nested loops join for the `part` table whereas the MAXDOP=1 plan uses hash join.

Therefore, performance evaluation models and resource estimation models for database workloads have to consider plan changes into consideration, along with other factors. From this study, it is clear that some queries are more parallelism-sensitive than others, and query plan shapes can highly vary with parallelism and scale factors. This insight can also be exploited for better resource allocation for queries, which could potentially lead to improved efficiency. Our analysis also emphasizes the need for integrated hardware-software strategies for resource management.

8. SENSITIVITY TO MEMORY CAPACITY

Queries make use of memory to store intermediate results and other temporary data during their execution. SQL Server estimates the memory requirement for every query (called the 'query memory grant' [26]) and reserves it at the start of execution, in order to improve reliability and prevent out-of-memory exceptions during execution. SQL Server also enforces a per-query maximum limit so as to ensure fairness.

Some queries require larger amounts of memory than others, depending upon the kind of physical operations involved in the query. For instance, a hash join typically requires more memory than a nested-loops join in order to build and maintain the hash table. The presence of blocking operators such as sort also influence the memory required by a query. The memory requirement also depends upon the degree of parallelism, usually requiring more memory if there are more parallel workers. For example, TPC-H query 20 uses 45% less memory with MAXDOP=1 compared to that with MAXDOP=32. However, a larger memory requirement by any query limits the concurrency that one can achieve on a given system.

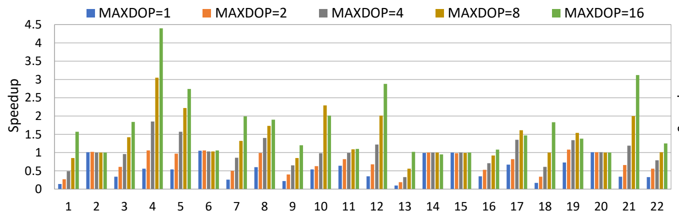
We studied the performance sensitivity to query memory limits for TPC-H with SF=100. This scale factor ensures that there is very less disk utilization, thereby isolating the sensitivity to memory. Figure 8 shows the execution time speedups for different query memory grant configurations (15%, 5%, and 2%) relative to the baseline default of 25% (approx. 9.2 GB on our system; of the total server memory, about 80% is allocated to SQL Server of which a portion is set aside for shared data structures, such as the buffer pool, and the rest is partitioned for per-query use).

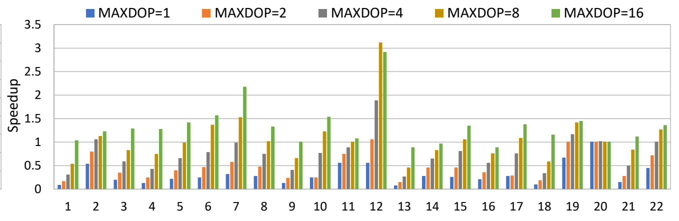
Except 7 queries (Q3, Q8, Q9 and Q13, Q16, Q18, Q21), the other queries are not very sensitive to memory capacity, since they do not degrade much with smaller memory. Q18 shows a high sensitivity, degrading at every configuration. Q21 and Q13 are able to tolerate the reduction of memory up to M=5% without impacting performance, and are only impacted at M=2%. By choosing appropriate query memory grants, more concurrent queries could be accommodated for a given total memory capacity, thereby increasing the overall concurrency of the database system. Thus, concurrency and sensitivity to memory capacity are closely related and should be studied together.

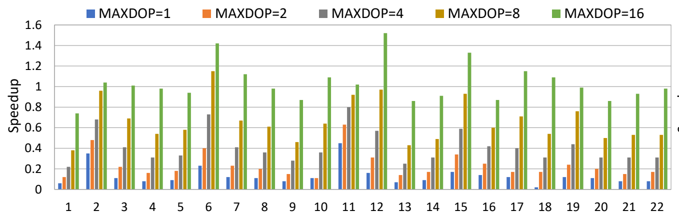
9. PERFORMANCE ANALYSIS PITFALLS

Having an appropriate experimental setup with proper data organization and a diverse workload suite is imperative for performance analysis of database systems. We recommend avoiding the following policies or antipatterns as they may lead to misleading conclusions or suboptimal management.

1. Evaluating current or future database server hardware efficiencies using a single class of workloads or single scale factors within each class of workloads.
2. Executing performance runs for analytical workloads with row-store storage format or for transactional workloads with column-store storage format.
3. Disregarding non-volatile storage bandwidth limitations on overall performance scalability while adding more cores or higher memory bandwidth.
4. Disregarding write bandwidth limitations on overall performance of transactional workloads even if the database fits within memory. While NVMe SSDs support good write bandwidths, it can be a bottleneck in servers hosting the database on hard disks.


(a) SF=10, Queries 1–22


(b) SF=30, Queries 1–22


(c) SF=100, Queries 1–22

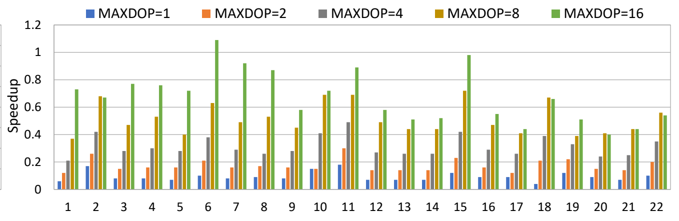
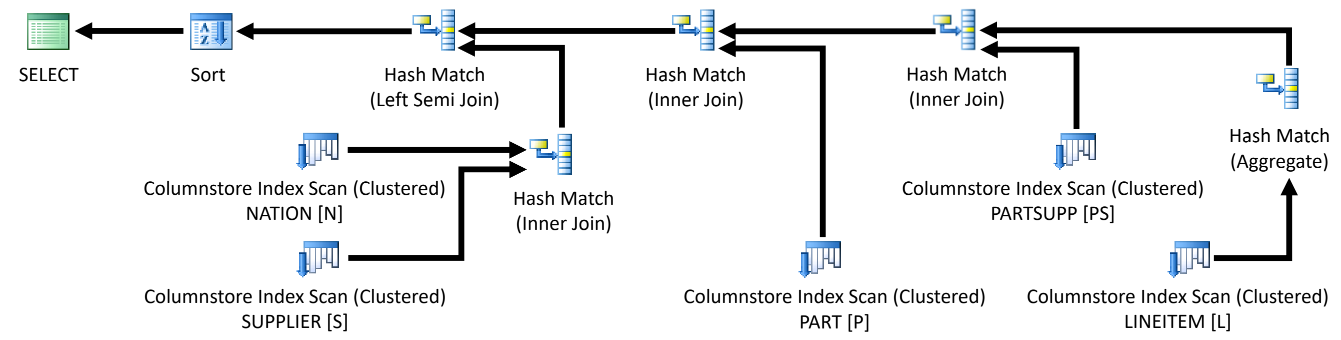
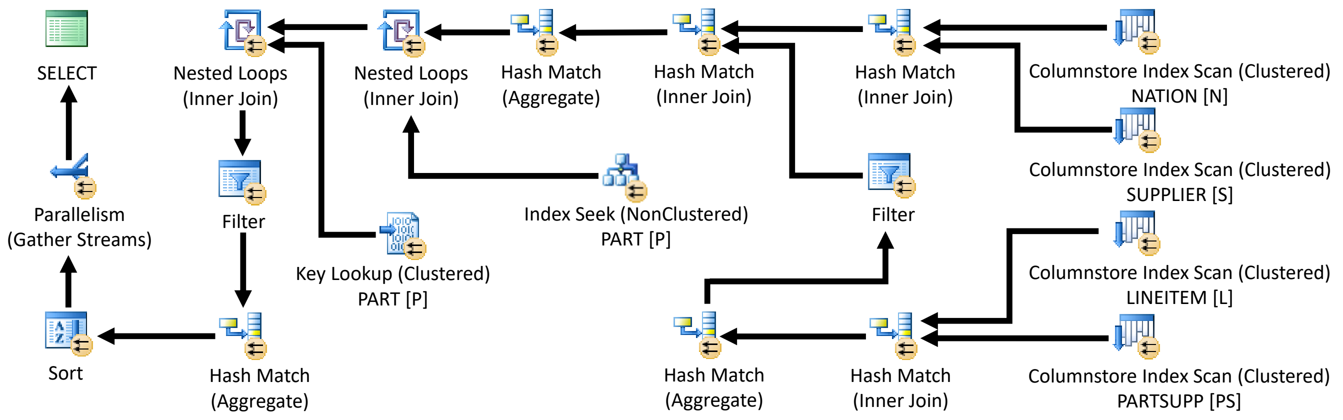

(d) SF=300, Queries 1–22

Figure 6: TPC-H query speedup, with limited MAXDOP and #cores, relative to the baseline with MAXDOP and #cores = 32.



(a) MAXDOP=1. Join order: $((N \bowtie S) \bowtie (P \bowtie (PS \bowtie L)))$.



(b) MAXDOP=32. Join order: $((((N \bowtie S) \bowtie (PS \bowtie L)) \bowtie P) \bowtie P)$.

Figure 7: Different query plans, for sequential and parallel execution, for TPC-H (SF=300) Query 20.

5. Separating maximum available parallelism/concurrency (depends on number of cores) and maximum available memory capacity as orthogonal aspects.

6. Being oblivious to alternate query plans while evaluating dynamic resource limitations.

7. Treating the DBMS as a black box for resource governance at the OS or hardware level. Modern commercial DBMSs can adapt query processing to available resources, for example, throttling parallelism in case of insufficient threads, so holistic hardware-software impact estimation and management are needed.

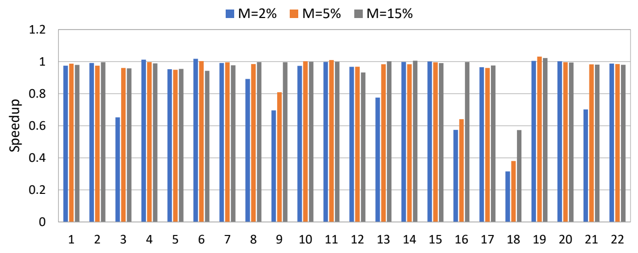


Figure 8: TPC-H (SF=100) execution time speedups for queries 1–22 with 2%, 5%, and 15% query memory grant relative to the baseline time with the default 25% query memory grant.

10. RESEARCH OPPORTUNITIES

Based on our observations in this paper, we highlight a few research questions having significant potential of benefitting DBaaS providers and database users.

- For a concurrent stream of query executions, how should system resources (cores, storage capacity, bandwidth) be partitioned among streams to meet SLAs? While core and bandwidth allocations can be quickly modified, runtime changes to query parallelism and allocated memory capacity have higher overheads as data may need to be repartitioned or spilled to secondary storage.
- Can a different query plan, along with a change in system resource allocation, improve performance or energy efficiency? What predictive models are needed to estimate impacts? Since the plan space is usually very large, it is infeasible to exhaustively analyze it at runtime. Cardinality dependence and pipelined execution of the query plan create additional challenges for performance models.
- In a concurrent stream, is it better to immediately start executing queries even with limited resources, or delay them till others finish and free up resources?
- Since scheduling on hyper-threaded cores can be beneficial or harmful, and the DBMS/workloads running within virtual machines have little/no control over scheduling on the host machine, how can core scheduling decisions in the cloud be coordinated between the host and the DBMS?
- Since transactional and analytical workloads exhibit different cache sensitivities, even a well-designed server running diverse database workloads will experience cache under-utilization. Can caches be dynamically reconfigured to use the excess capacity for other purposes?

11. RELATED WORK

A number of performance analysis and characterization studies of database workloads have focused on single classes of workloads, e.g., only OLTP workloads [1–7, 34–36] or only DSS workloads [8, 9, 37]. Like some other works [38–41] we consider both OLTP and DSS workloads, but differ from them in demonstrating changing resource sensitivities with different database scale factors for each class. Additionally, we also evaluate hybrid (HTAP) class of workloads.

While studying database workloads, researchers often analyze stall times and execution time breakdowns to identify workload performance bottlenecks in the system under test [7, 41]. For example, Sirin et al. [7] observed that stall cycles due to L1I cache misses constitute more than half of the execution cycles and that in-memory OLTP systems behave similar to disk-based systems. While stall times provide insight into workload characteristics with the current resource settings, they do not provide guidance on optimal resource allocations for the current or future systems. Our work aims to address this gap. Our study also aids efforts towards achieving predictable performance on database workloads [42, 43].

Computer architects have used CAT [27] to study and guide hardware cache partitioning decisions but have not extensively studied database workloads in this context. Cook et al. [32] evaluated partitioning and co-scheduling policies with SPEC CPU2006, DaCapo, and Parsec benchmark suites, and microbenchmarks. Like us, they conclude that the LLC is over-provisioned but do not observe knees in execution time profiles for most applications. Our experiments show a sharp degradation in performance below a certain LLC allocation size. Thus, cache partitioning strategies that exploit knowledge of the working sets or knees [44–46] could be useful with the workloads we study. Lo et al. [47] used CAT to provide LLC isolation for websearch, machine learning, and key-value store applications.

Other approaches to studying cache sensitivity include making use of full-system simulation [2] and analytical models [48] but may have limitations in the modeling accuracy of the real systems they study. Keeton et al. [1] physically swapped processor/cache boards to study the performance impact with different L2 cache sizes, but this approach is limited in feasibility by the number of configurations that can be studied.

Prior works [49, 50] have pointed out that hyper-threading can be beneficial or detrimental to performance depending on shared resource interference. We show that for database workloads, this can vary widely depending on the database size, as computation and I/O requirements change, even within the same class of workloads.

12. CONCLUSIONS

In this paper we study the performance sensitivity of transactional, analytical, and hybrid database workloads to server resources for Microsoft SQL Server running on Linux. Our work reinforces the view that modern server hardware is not optimally provisioned for running database workloads. Last-level caches and memory bandwidths appear to be underutilized whereas increasing the number of cores in conjunction with non-volatile storage bandwidths can be beneficial. We also show that resource sensitivities vary with the workload class and data size, so some over-provisioning at design time may be unavoidable. Future research can explore improving runtime efficiencies with better scheduling policies and coordinated hardware-software governance that automatically adapt query processing to dynamically changing levels of user/query concurrency and resource availability.

13. ACKNOWLEDGMENTS

The authors thank Alan Halverson, Jignesh Patel, Mike Zwilling, and anonymous reviewers for helpful comments.

14. REFERENCES

[1] K. Keeton, D. A. Patterson, Y. Q. He, R. C. Raphael, and W. E. Baker, "Performance characterization of a quad Pentium Pro SMP using OLTP workloads," in *Proceedings of the 25th Annual International Symposium on Computer Architecture*, ISCA '98, (Washington, DC, USA), pp. 15–26, IEEE Computer Society, 1998.

[2] R. Stets, K. Gharachorloo, and L. A. Barroso, "A detailed comparison of two transaction processing workloads," in *IEEE International Workshop on Workload Characterization*, pp. 37–48, Nov 2002.

[3] A. R. Alameldeen, M. M. K. Martin, C. J. Mauer, K. E. Moore, M. Xu, M. D. Hill, D. A. Wood, and D. J. Sorin, "Simulating a $2m commercial server on a $2k PC," *Computer*, vol. 36, pp. 50–57, Feb. 2003.

[4] R. A. Hankins, T. Diep, M. Annavaram, B. Hirano, H. Eri, H. Nueckel, and J. P. Shen, "Scaling and characterizing database workloads: Bridging the gap between research and practice," in *Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 36, (Washington, DC, USA), pp. 151–162, IEEE Computer Society, 2003.

[5] P. Tözün, I. Pandis, C. Kaynak, D. Jevdjic, and A. Ailamaki, "From A to E: Analyzing TPC's OLTP benchmarks: The obsolete, the ubiquitous, the unexplored," in *Proceedings of the 16th International Conference on Extending Database Technology*, EDBT '13, (New York, NY, USA), pp. 17–28, ACM, 2013.

[6] P. Tözün, B. Gold, and A. Ailamaki, "OLTP in wonderland: Where do cache misses come from in major OLTP components?," in *Proceedings of the Ninth International Workshop on Data Management on New Hardware*, DaMoN '13, (New York, NY, USA), pp. 8:1–8:6, ACM, 2013.

[7] U. Sirin, P. Tözün, D. Porobic, and A. Ailamaki, "Micro-architectural analysis of in-memory OLTP," in *Proceedings of the 2016 International Conference on Management of Data*, SIGMOD '16, (New York, NY, USA), pp. 387–402, ACM, 2016.

[8] P. Trancoso, J.-L. Larriba-Pey, Z. Zhang, and J. Torrellas, "The memory performance of DSS commercial workloads in shared-memory multiprocessors," in *Proceedings of the 3rd IEEE Symposium on High-Performance Computer Architecture*, HPCA '97, (Washington, DC, USA), pp. 250–260, IEEE Computer Society, 1997.

[9] R. Yu, L. Bhuyan, and R. Iyer, "Comparing the memory system performance of DSS workloads on the HP V-Class and SGI Origin 2000," in *Proceedings of the 16th International Symposium on Parallel and Distributed Processing*, IPDPS '02, (Washington, DC, USA), pp. 31–36, IEEE Computer Society, April 2002.

[10] A. Kemper and T. Neumann, "Hyper: A hybrid OLTP&OLAP main memory database system based on virtual memory snapshots," in *Proceedings of the 2011 IEEE 27th International Conference on Data Engineering*, ICDE '11, (Washington, DC, USA), pp. 195–206, IEEE Computer Society, 2011.

[11] J. Arulraj, A. Pavlo, and P. Menon, "Bridging the archipelago between row-stores and column-stores for hybrid workloads," in *Proceedings of the 2016 International Conference on Management of Data*, SIGMOD '16, (New York, NY, USA), pp. 583–598, ACM, 2016.

[12] R. Appuswamy, M. Karpathiotakis, D. Porobic, and A. Ailamaki, "The case for heterogeneous HTAP," in *Conference on Innovative Data Systems Research*, 2017.

[13] A. Ailamaki, "The next 700 transaction processing engines," in *Proceedings of the 2017 ACM International Conference on Management of Data*, SIGMOD '17, (New York, NY, USA), pp. 1–2, ACM, 2017.

[14] "The TPC-E benchmark." `http://www.tpc.org/tpce/`.

[15] "Azure SQL database benchmark overview." `https://docs.microsoft.com/en-us/azure/sql-database/sql-database-benchmark-overview`, 2016.

[16] "Azure SQL database." `https://azure.microsoft.com/en-us/services/sql-database/`, 2017.

[17] "SQL database options and performance: Understand what's available in each service tier." `https://docs.microsoft.com/en-us/azure/sql-database/sql-database-service-tiers`, 2017.

[18] "Transactional (OLTP) - a technical reference guide for designing mission-critical OLTP solutions." `https://technet.microsoft.com/en-us/library/hh393539(v=sql.110).aspx`.

[19] S. Chaudhuri and U. Dayal, "Data warehousing and OLAP for decision support," in *Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data*, SIGMOD '97, (New York, NY, USA), pp. 507–508, ACM, 1997.

[20] "Symmetric multi-processing (DW) - a technical reference guide for designing mission-critical DW solutions." `https://technet.microsoft.com/en-us/library/hh393544(v=sql.110).aspx`.

[21] "The TPC-H benchmark." `http://www.tpc.org/tpch/`.

[22] D. J. Abadi, S. R. Madden, and N. Hachem, "Column-stores vs. row-stores: How different are they really?," in *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, SIGMOD '08, (New York, NY, USA), pp. 967–980, ACM, 2008.

[23] "Get started with columnstore for real time operational analytics." `https://docs.microsoft.com/en-us/sql/relational-databases/indexes/get-started-with-columnstore-for-real-time-operational-analytics`.

[24] "SQL Server on Linux documentation." `https://docs.microsoft.com/en-us/sql/linux/`, 2017.

[25] "Processor counter monitor (PCM)." `https://github.com/opcm/pcm`, 2017.

[26] K. Delaney, B. Beuchemin, and C. Cunningham., *Microsoft SQL Server 2012 Internals*. Microsoft Corporation, 2013.

[27] Intel Corporation, "Improving real-time performance by utilizing cache allocation technology," Apr. 2015.

[28] "PQoS/Intel(R) RDT utility." `https://github.com/01org/intel-cmt-cat/tree/master/pqos`, 2017.

[29] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta, "The SPLASH-2 programs: Characterization and methodological considerations," in *Proceedings of the 22Nd Annual International Symposium on Computer Architecture*, ISCA '95, (New York, NY, USA), pp. 24–36, ACM, 1995.

[30] M. Ferdman, A. Adileh, O. Kocberber, S. Volos, M. Alisafaee, D. Jevdjic, C. Kaynak, A. D. Popescu, A. Ailamaki, and B. Falsafi, "Clearing the clouds: A study of emerging scale-out workloads on modern hardware," in *Proceedings of the Seventeenth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XVII, (New York, NY, USA), pp. 37–48, ACM, 2012.

[31] P. Lotfi-Kamran, B. Grot, M. Ferdman, S. Volos, O. Kocberber, J. Picorel, A. Adileh, D. Jevdjic, S. Idgunji, E. Ozer, and B. Falsafi, "Scale-out processors," in *Proceedings of the 39th Annual International Symposium on Computer Architecture*, ISCA '12, (Washington, DC, USA), pp. 500–511, IEEE Computer Society, 2012.

[32] H. Cook, M. Moreto, S. Bird, K. Dao, D. A. Patterson, and K. Asanovic, "A hardware evaluation of cache partitioning to improve utilization and energy-efficiency while preserving responsiveness," in *Proceedings of the 40th Annual International Symposium on Computer Architecture*, ISCA '13, (New York, NY, USA), pp. 308–319, ACM, 2013.

[33] "Block IO controller." `https://www.kernel.org/doc/Documentation/cgroup-v1/blkio-controller.txt`, 2017.

[34] S. Harizopoulos and A. Ailamaki, "Improving instruction cache performance in OLTP," *ACM Trans. Database Syst.*, vol. 31, pp. 887–920, Sept. 2006.

[35] S. Harizopoulos, D. J. Abadi, S. Madden, and M. Stonebraker, "OLTP through the looking glass, and what we found there," in *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, SIGMOD '08, (New York, NY, USA), pp. 981–992, ACM, 2008.

[36] D. Porobic, I. Pandis, M. Branco, P. Tözün, and A. Ailamaki, "OLTP on hardware islands," *Proceedings of the VLDB Endowment*, vol. 5, pp. 1447–1458, July 2012.

[37] P. Vaidya and J. J. Lee, “Characterization of TPC-H queries for a column-oriented database on a dual-core AMD Athlon processor,” in *Proceedings of the 17th ACM Conference on Information and Knowledge Management*, CIKM ’08, (New York, NY, USA), pp. 1411–1412, ACM, 2008.

[38] L. A. Barroso, K. Gharachorloo, and E. Bugnion, “Memory system characterization of commercial workloads,” in *Proceedings of the 25th Annual International Symposium on Computer Architecture*, ISCA ’98, (Washington, DC, USA), pp. 3–14, IEEE Computer Society, 1998.

[39] P. Ranganathan, K. Gharachorloo, S. V. Adve, and L. A. Barroso, “Performance of database workloads on shared-memory systems with out-of-order processors,” in *Proceedings of the Eighth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS VIII, (New York, NY, USA), pp. 307–318, ACM, 1998.

[40] J. L. Lo, L. A. Barroso, S. J. Eggers, K. Gharachorloo, H. M. Levy, and S. S. Parekh, “An analysis of database workload performance on simultaneous multithreaded processors,” in *Proceedings of the 25th Annual International Symposium on Computer Architecture*, ISCA ’98, (Washington, DC, USA), pp. 39–50, IEEE Computer Society, 1998.

[41] A. Ailamaki, D. J. DeWitt, M. D. Hill, and D. A. Wood, “DBMSs on a modern processor: Where does time go?,” in *Proceedings of the 25th International Conference on Very Large Data Bases*, VLDB ’99, (San Francisco, CA, USA), pp. 266–277, Morgan Kaufmann Publishers Inc., 1999.

[42] P. Unterbrunner, G. Giannikis, G. Alonso, D. Fauser, and D. Kossmann, “Predictable performance for unpredictable workloads,” *Proceedings of the VLDB Endowment*, vol. 2, pp. 706–717, Aug. 2009.

[43] J. Huang, B. Mozafari, G. Schoenebeck, and T. F. Wenisch, “A top-down approach to achieving performance predictability in database systems,” in *Proceedings of the 2017 ACM International Conference on Management of Data*, SIGMOD ’17, (New York, NY, USA), pp. 745–758, ACM, 2017.

[44] G. E. Suh, S. Devadas, and L. Rudolph, “A new memory monitoring scheme for memory-aware scheduling and partitioning,” in *Proceedings of the 8th International Symposium on High-Performance Computer Architecture*, HPCA ’02, (Washington, DC, USA), pp. 117–128, IEEE Computer Society, 2002.

[45] M. K. Qureshi and Y. N. Patt, “Utility-based cache partitioning: A low-overhead, high-performance, runtime mechanism to partition shared caches,” in *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 39, (Washington, DC, USA), pp. 423–432, IEEE Computer Society, 2006.

[46] A. Pan and V. S. Pai, “Imbalanced cache partitioning for balanced data-parallel programs,” in *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-46, (New York, NY, USA), pp. 297–309, ACM, 2013.

[47] D. Lo, L. Cheng, R. Govindaraju, P. Ranganathan, and C. Kozyrakis, “Heracles: Improving resource efficiency at scale,” in *Proceedings of the 42Nd Annual International Symposium on Computer Architecture*, ISCA ’15, (New York, NY, USA), pp. 450–462, ACM, 2015.

[48] R. Sen and D. A. Wood, “Reuse-based online models for caches,” in *Proceedings of the ACM SIGMETRICS/International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS ’13, (New York, NY, USA), pp. 279–292, ACM, 2013.

[49] C. Jung, D. Lim, J. Lee, and S. Han, “Adaptive execution techniques for SMT multiprocessor architectures,” in *Proceedings of the Tenth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, PPoPP ’05, (New York, NY, USA), pp. 236–246, ACM, 2005.

[50] M. Curtis-Maury and T. Wang, “Integrating multiple forms of multithreaded execution on multi-SMT systems: A study with scientific applications,” in *Proceedings of the Second International Conference on the Quantitative Evaluation of Systems*, QEST ’05, (Washington, DC, USA), pp. 199–208, IEEE Computer Society, 2005.