Paper

Performance Modeling of Database Systems: a Survey

Antonina Krajewska

Institute of Control and Computation Engineering, Warsaw University of Technology, Warsaw, Poland

https://doi.org/10.26636/jtit.2018.128918

Abstract—This paper presents a systematic survey of the existing database system performance evaluation models based on the queueing theory. The continuous evolution of the methodologies developed is classified according to the mathematical modeling language used. This survey covers formal models – from queueing systems and queueing networks to queueing Petri nets. Some fundamentals of the queueing system theory are presented and queueing system models are classified according to service time distribution. The paper introduces queueing networks and considers several classification criteria applicable to such models. This survey distinguishes methodologies, which evaluate database performance at the integrated system level. Finally, queueing Petri nets are introduced, which combine modeling power of queueing networks and Petri nets. Two performance models within this formalism are investigated. We find that an insufficient amount of research effort is directed into the area of NoSQL data stores. Vast majority of models developed focus on traditional relational models. These models should be adapted to evaluate performance of non-relational data stores.

Keywords*—*database systems, NoSQL data stores, performance evaluation, queueing networks, queueing Petri nets.

1. Introduction

Database servers play a crucial role in information system infrastructures. With the rapid expansion of Big Data analytics, NoSQL data stores keep gaining strategic significance and supplement traditional relational databases. Extensive research has been conducted in the area of performance of relational databases, which has been reviewed thoroughly [1], [2]. The vast majority of existing models is based on the queueing theory. Still, the performance of NoSQL data stores remains unexplored. Therefore, existing performance models should be revisited to identify their potential in capturing the dynamics of non-relational systems. This paper presents a survey of the existing database performance models with their underlying queueing networks and queueing Petri nets.

Several classification criteria may be considered in the analysis of the performance models constructed. Existing models may be categorized according to the field of study, i.e. concurrency control, replication mechanism or database architecture. Depending on the phenomena investigated, researchers may choose different performance criteria, such as request response time or transaction throughput. Finally, analytical and simulation studies may be distinguished.

In this paper, existing models are classified according to their mathematical modeling formalism. Firstly, models which are based on the queueing system are considered, then a review of the queueing network models is presented. Lastly, the paper presents queueing Petri nets as a powerful tool for qualitative and quantitative analysis of the performance of database systems. With this approach adopted, the article demonstrates the evolution of performance evaluation models. Section 2 presents fundamentals of the queueing systems theory and a review of queueing system-based representation of database dynamics. In Section 3, queueing network models are considered and various aspects of such models, including the open and closed character of a network or service time distribution, are recorded. Following Osman and Knottenbelt, we also classify them according to the granularity of transaction modeling and describe two performance evaluation models which enable the mapping of database design specification at the integrated system level to the queueing network structure [2]. Finally, in Section 4, queueing Petri nets are introduced which combine the expressiveness of modeling of queueing networks and Petri nets.

2. Queueing Systems

This section presents database performance evaluation models with a single underlying queueing system node. A queueing system is defined by the following: (i) stochastic process describing the customer arrival stream $A(t)$, (ii) probability distribution of service time $B(t)$ per customer, (iii) number of service channels K and (iv) scheduling discipline in the queue, i.e. first-come-first-serve (FCFS), last-come-first-serve (LCFS), processor sharing (PS), round robin (RR), etc. [3]. In this paper, the simple Kendall's notation is used: $A/B/K$ [3]. If no other scheduling discipline is denoted, the FCFS variety is adopted. The queueing theory enables a probabilistic analysis of such systems considering the average length, probability that the queue has a given length, or average service time. The relation between the average number of customers in a system is given by applying Little's law. It claims that the average number of customers in a queueing system is equal to the average arrival rate of customers to that system, times the average time spent in that system. Queueing system models are capable of system-level workload evaluation for both centralized and distributed database systems [3]. This section focuses on Markovian models and classification according to the service time distribution.

2.1. Markovian Models with Exponentially Distributed Service Times

Firstly, we will discuss $M/M/m$ models in which customers arrive at rate λ, according to the Poisson process, and are served by m servers. The service time for each customer has an exponential distribution with parameter μ. The $M/M/1$ system with one server (Fig. 1) is a special kind of a Markovian model with exponentially distributed service time.

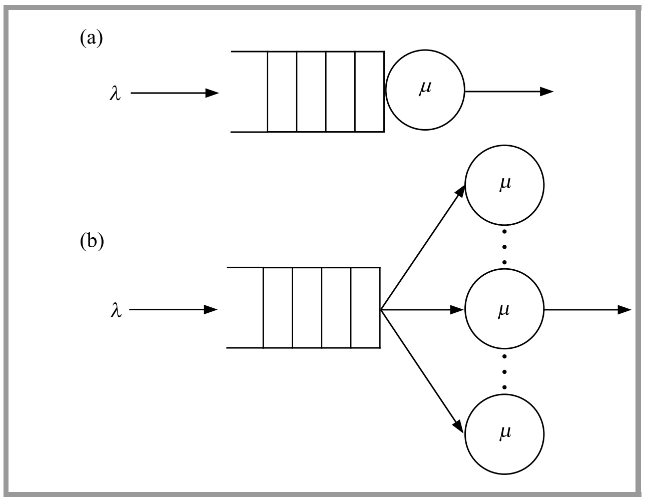

Fig. 1. Queueing system: (a) $M/M/1$, (b) $M/M/m$.

Let $N(t)$ denote the number of customers in a system at time t, then $N(t)$ is a continuous time Markov chain shown in Fig. 2.

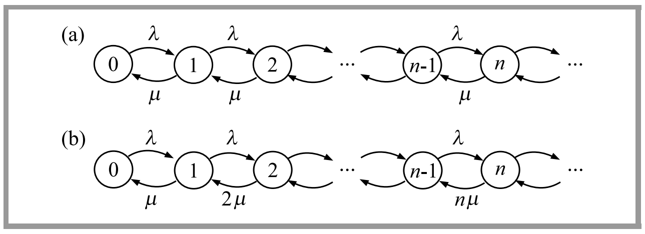

Fig. 2. State space diagram for the Markov chain for: (a) $M/M/1$, (b) $M/M/m$ systems.

Nicola and Jarke in [1] review the performance models of distributed and replicated database systems. They record that first queueing models of distributed databases were constructed by Coffmann *et al.* [4], Bacelli and Coffman [5], as well as Nelson and Iyer [6]. Nelson and Iyer [6] compare the performance of synchronous and non-synchronous updating policies in the $M/M/m$ system with non-preemptive processing of write operations. By contrast, Bacelli and Coffmann *et al.* [5] analyze different replication policies in a system with preemptive priority for write requests over read requests. Interruptions dynamics was captured by the $M/M/1$ system in the following way: read transactions could occupy m servers concurrently, until write requests arrived with the rate λ, according to the Poisson process, and were processed by all of the m servers with exponentially distributed service time.

2.2. Markovian Models with Generally Distributed Service Times

In a $M/G/1$, queueing system customers arrive according to the Poisson distribution process and have generally distributed service times. In contrast to $M/M/1$ queues, general models do not a have closed form describing the number of jobs in the system in a stationary state. However, according to the Pollaczek-Khinchine formula, the average $M/G/1$ queue length is given by Eq. 1 [3]:

$$L = \rho + \frac{\rho^2 + \lambda^2 \mathrm{Var}(S)}{2(1-\rho)} \ , \tag{1}$$

where λ is the arrival rate of the Poisson process, $\frac{1}{\mu}$ is the mean of the service time distribution S, $\rho = \frac{\lambda}{\mu}$ is the utilization and $\mathrm{Var}(S)$ is the variance of the service time distribution S.

Arzauga and Kaeli [7] construct $M/G/1$ model of a storage area network (SAN) system that manages multiple applications stored in the same volume. Their experiments included various types of workloads running in the same system.

2.3. Load Dependent Systems M/M/m-LDS

Kihl *et al.* [8] have shown that the $M/M/1$ model does not capture the high load dynamics of a database for the write operations workload. Paper [9] proposes a model which adds load dependency to the service time. Its authors construct the *M/M/m-LDS* model in which service time depends on the number of concurrent requests.
Let $x_t(n)$ be the service time at time t, and n be the number of concurrent requests in the system. Then:

$$x_t(n) = x_{base}(1+f)^{n-1} \ , \tag{2}$$

where x_{base} is exponentially distributed base service time for a system containing one job and $f \in [0,1]$ is a dependency factor. The model has been validated with experimental data and its accuracy in predicting response time for a write-heavy workload has been proved. Figure 3 presents

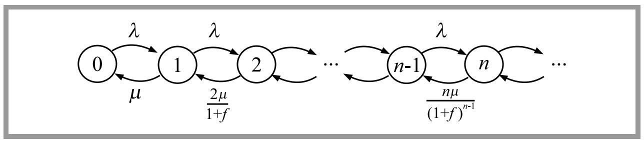

Fig. 3. State space diagram for Markov chain for the *M/M/m-LDS* system.

Markov chain for $N(t)$ process of the number of customers in the $M/M/m$-*LDS* system.

3. Queueing Networks

Database models based on single queueing systems can capture only particular aspects of database system dynamics. The authors of [1] have noted disadvantages of this approach in modeling replicated and distributed database systems, such as using the same queue by all database sites or neglecting inter-site communication. Furthermore, in the case of replicated database systems, full replication must be assumed. These limitations do not have to apply to queueing network models [3]. Queueing networks consist of several queueing nodes. Customers are routed between the nodes probabilistically. After being serviced at one queue node, the customer may join another node to receive additional service or may depart the network [3].

3.1. Open and Closed Queueing Networks

Open queueing networks allow external arrivals of customers. The number of customers in the network is variable and the arrival rate does not depend on the number of currently processed requests. To the contrary, *closed queueing networks* consider s fixed number of customers in the system. Once a request has been served, it is replaced with another one.

Open queueing networks have been widely used in performance modeling of database systems [10]–[13]. In [14], Mei *et al.* represent a database with an open two-node queueing network with a central processor-sharing node and multiple multi-server nodes. Parallel access to the backend database is modeled as a multi-service FCFS queue with exponential service times.

Since the number of requests arriving at the database system is not fixed, open queueing networks are generally more adequate. However, in specific problems, closed networks simplify performance evaluation. Nicola and Jarke record their significance in an analysis of concurrency control mechanisms. Performance of such methods depends on the multiprogramming level and thus its evaluation becomes challenging in networks with a variable number of jobs [1].

Liang and Tripathi in [15] introduce saga transactions – a special case of long-lived transactions (LLTs) – which release their locks as soon as possible. To evaluate the performance of saga systems, authors define an analytical model with a closed queueing network underlying. Carey and Livini [16] deploy a closed queueing network to analyze the performance of different concurrency control mechanisms in distributed database systems.

3.2. Service Time Distribution

Exponentially distributed service time. The simplest networks consist of $M/M/1$ queues, thus service time of all transactions is distributed according to the same exponential distribution. Ciciani *et al.* [17], [18] develop an analytical model to compare concurrency control in a replicated, distributed environments. In the queueing network, each database site is represented as an $M/M/1$ queue.

Generally distributed service time. More general models use $M/G/1$ queues. Banerjee *et al.* [19] use networks of $M/G/1$ queue nodes to benchmark the concurrency control protocol developed in a distributed environment against existing concurrency control schemes. Hwang *et al.* [20] compare the performance of three replication schemes using a model in which each database site is represented by an $M/G/1/RR$ node with the round robin scheduling discipline.

$\boldsymbol{M/H_n/*}$ **models**. The above-mentioned models do not distinguish transactions according to service time distribution. As a remedy for this unrealistic assumption, networks with $M/H_n/*$ queue nodes might be used. In such models, arriving requests are divided into n categories with different service times. Since the service time in each class has exponential distribution, service time for the combined arrival process follows an n-phase hyper-exponential distribution. Leung [21] assigns different exponentially distributed service times for read and update requests. He uses a network with $M/H_2/1$ queueing nodes with 2-phase hyper-exponentially distributed service times.

Nicola and Jarke [1] introduce an analytical model which emphasizes the mutual influence between replication and inter-site communication. Each database site is represented by an $M/H_n/1$ queueing node.

Deterministic service time. Born in [22] uses an $M/D/1$ queueing network to investigate trade-offs of different implementation strategies for distributed lock management. The algorithms compared differ in the management of the lock database, in its optimization and in the communication protocol.

3.3. Granularity of Transaction Modeling

Osman and Knottenbelt [2] classify queueing database performance models according to the granularity of transaction modeling. In their exhaustive survey, the authors distinguish four types of performance evaluation models: *black box, transaction processing, transaction size and transaction phase*.

Black box models. Such models categorize transactions according to their service demands at the system level.

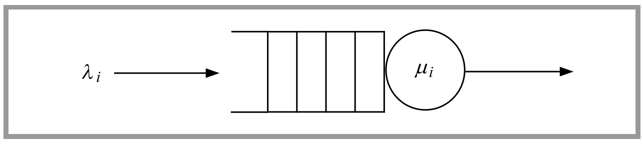

Fig. 4. Black box model: transaction class i is defined by arrival rate λ_i and service rate μ_i.

Transaction T_i from class i arrives to the database with the rate λ_i and a service rate μ_i (Fig. 4).

Black box models may represent both centralized and distributed database systems. The former design type is represented by a queueing system, whereas the latter one can be modeled both with a queueing system with multiple servers [5], [6] or as a queueing network with multiple nodes [17], [18]. Such a model can vary in service time distribution as discussed in the previous subsection.

Transaction processing models. In transaction processing models, the queueing network represents hardware architecture which lies below the database system. Transactions arriving to the system are classified according to their service demands on particular components of the hardware architecture, for example CPU or disk resources (Fig. 5). For each transaction class T_i arrival rate λ_i is defined. Transactions are routed throughout the network probabilistically.

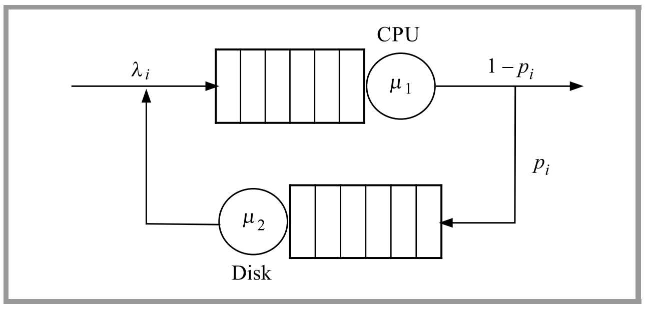

Fig. 5. Transaction processing model. Transaction class i is defined by arrival rate λ_i and resource demands: CPU service rate μ_1 and disk service rate μ_2. Transactions are routed between service stations probabilistically.

In the transaction processing approach, the main variables to be optimized are capacity or quantity of physical resources [2].

Menasce *et al.* [23] emphasize the impact of considering both software and hardware contention. Software contention is caused by a limited number of threads that may process arriving transactions. When all threads in the system are busy, requests are placed in a queue. The number of active threads is defined by a birth-death process. Hardware contention is captured by a closed queueing network with nodes representing CPU and disk resources. Authors study both single- and multi-class requests categorization.

Gijsen *et al.* [24] study sojourn times in an open queueing network with a single processor sharing node and an arbitrary number of multi-server FCFS nodes. PS-node represents the front-end application server, while multi-server FCFS nodes stand for distributed database sites. The Poisson process defines the processor sharing node's arrival stream. After being processed by the front-end node, the request is routed to one of a database site nodes or departs the system. After customer has been serviced in one of FCFS nodes, it jumps back to a processor sharing node.

Transaction size model. In this approach, transaction class T_i is defined by the number n_i of data objects it accesses. Osman and Knottenbelt in [2] mention various object types including table rows, data pages or locks. Resource demands on hardware resources depend on n (see Fig. 6).

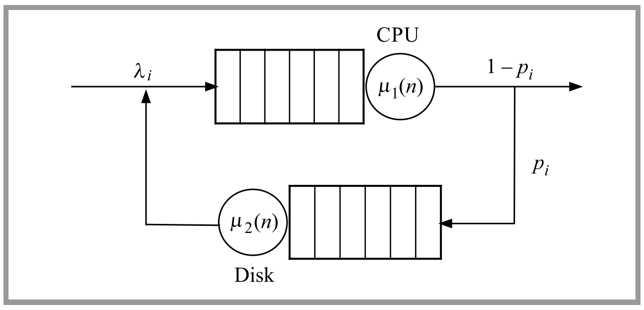

Fig. 6. Transaction size model. Transaction class i is defined by arrival rate λ_i and number of objects n it accesses. Resource demands depends on n: CPU service rate $\mu_1(n)$ and disk service rate $\mu_2(n)$.

Thomasian and Ryu [25] develop a model predicting the maximum throughput of both locking and optimistic concurrency control algorithms in a centralized database environment. Each transaction class accesses a fixed number of randomly chosen data objects. This set is then used to determine locking conflict probability. A closed queueing network represents the hardware architecture of a system with processor sharing and disks queues. Authors define the lock scheduling overhead as a function of the number of idle and active transactions. The same approach has been used by Morris and Wong [26]. However, the authors neglected the concurrency overhead.

Transaction phase. In the transaction phase model, transactions are categorized according to execution phases they consist of. Phases might be parallel or sequential. Queue nodes in a network represent these phases and transactions are routed between them probabilistically (see Fig. 7).

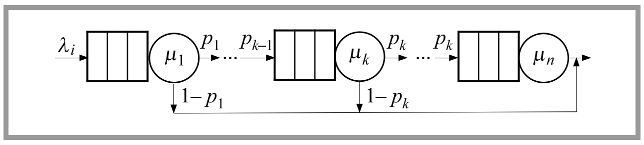

Fig. 7. Transaction phase model. Transaction class i is defined by arrival rate λ_i and number of phases n (p_i – probability of moving from phase i to phase $i+1$ and μ_i service rate in phase i).

Yu *et al.* [27] distinguish three processing phases: *front-end*, *application* and *database request*. In their study, the authors investigate dynamic transaction routing in locally distributed databases.

3.4. Database System Performance Evaluation Models

Following Osman and Knottenbelt [2], we would like to describe performance analysis methodologies which do not focus on particular database management system (DBMS) components or constructs, but evaluate DBMS performance of the integrated system. Such an approach allows mapping database system specifications onto queueing network models. In their survey shown in paper [2], the authors present eight methodologies. In this paper, attention is focused on the two most recent of them.

Parallel relational database system performance evaluation. Tomov *et al.* [28] describe an analytical methodology for response time estimation. The queries analyzed execute within a shared-nothing parallel DBMS. The proposed approach consists of three steps: *preparation*, *mean resource time estimation* and *mean query response time estimation*. In the first step, the query is transformed into a *query resource profile*. Via an execution plan transaction, it is reduced to a set of low-level resource usage specifications which determines its demands for hardware components. In the second stage, response time of particular resources is estimated. Each hardware resource is a $M/M/1$ or $M/M/G$ node in the queueing network. Synchronization between query execution phases, including pipelined execution or partitioned parallelism, is not taken into consideration during this stage. At the last stage, the mean time of a query is estimated by accommodating hardware resource times for the entire query usage profile. In this stage, intra-operator parallelism, such as pipelined or partitioned execution, determines the way usage time is accumulated.

QuePED model. Osman *et al.* [29] define a database design as a set of tables and transaction types accessing these tables. Authors introduce QuePED – a queueing network performance evaluation model for database designs. Each table in the database is represented as a system of queues in a network. It is noted that partitioned or replicated tables are treated as separate nodes. Each table is characterized by the following qualities: the attribute data types and selectivity, the expected number of rows and row length, as well as index type and structure. Customers arriving to the network correspond to transactions and are categorized according to their service demands in terms of the number of I/O pages required to process the transaction. To obtain this value, a query optimizer is used to return an optimal execution plan comprising SQL statements. Then, for each SQL statement, I/O cost is calculated with the approach described by Ramakrishnan and Gehrke [30]. Thus, the cost depends on the file structure of the table, for example heap file with no index, sorted file, cluster B+ tree file, and the type of SQL operation, such as scan, equality or range search, insert, update or delete. QuePED was validated with a TPC-C benchmark. Since the model assumes the mean time for read/write page requests, the *strace* utility was used to measure the time in which the kernel fulfilled a database page request.

4. Queueing Petri Nets

Queueing Petri Net (QPN) models combine queueing networks and Petri nets formalism and were introduced by Bause [31]. They extend Petri nets with queueing places which consist of two parts: the queue and the depository. Tokens arriving to the queueing place are firstly placed in a queue according to the scheduling strategy of the queue server. After being serviced, each token is placed in a depository for future transitions. Bause [31] distinguishes two types of queueing places. In *timed queueing*, service time distribution is given, while in *immediate queueing places*, the scheduling strategy is implemented without any delay. The motivation behind QPN is to fuse the modeling power of quantitative and qualitative analysis provided by queueing networks and Petri nets, respectively. Several other advantages were listed by Kounev *et al.* [32], such as ability to model simultaneous resource possession, synchronization, asynchronous processing and software contention. These models may capture hardware- and software-related aspects of system behavior. Moreover, the graphical representation of QPN is intuitive.

4.1. Notation and Definitions

Following Kounev *et al.* [32], we will now provide a formal definition of QPN.

Definition 1: A queueing Petri net is an 8-tuple $QPN = (P, T, C, I^+, I^-, M_0, Q, W)$ where:

1. $P = p_1, \ldots, p_n$ is a finite and non-empty set of places;
2. $T = t_1, \ldots, p_m$ is a finite and non-empty set of transitions, $P \cap T = \emptyset$;
3. C is a color function that assigns a finite and non-empty set of colors to each place and a finite and non-empty set of modes to each transitions;
4. I^+ and I^- and forward and backward incidence functions defined on $P \times T$, such that $I^-(p,t)$, $I^+(p,t) \in [C(t) \rightarrow C(p)_{MS}]$, $\forall (p,t) \in P \times T$, where $C(p)_{MS}$ denotes the set of all finite multisets of $C(p)$;
5. M_0 is a function on P describing *initial marking* such that $M_0(p) \in C(p)_{MS}$;
6. $Q = \big(Q_1, Q_2, (q_1, \ldots, q_{|P|})\big)$ where:
   - $Q_1 \subseteq P$ is the set of timed queueing places,
   - $Q_2 \subseteq P$ is the set of immediate queueing places,
   - $Q_1 \cap Q_2 = \emptyset$,
   - q_i denotes the description of a queue taking all colors of $C(p_i)$ into consideration if p_i is a queueing place or equals keyword 'null', if p_i is an ordinary place;
7. $W = \big(W_1, W_2, (w_1, \ldots, w_{|T|})\big)$ where:
   - $W_1 \subseteq T$ is the set of timed transitions,
   - $W_2 \subseteq T$ is the set of immediate transitions,
   - $T = W_1 \cup W_2, W_1 \cap W_2 = \emptyset$ and

- $w_i \in [C(t_i) \to \mathbb{R}^+]$, $\forall t_i \in T$, $c \in C(t_i)$ $w_i(c) \in R^+$ is interpreted as a rate of a negative exponential distribution specifying the firing delay due to color c, if $t_i \in W_1$ or a firing weight specifying the relative firing frequency due to color c, if $t_i \in W_2$.

A more detailed information about QPN may be found in [31].

4.2. Cassandra Replication Modeling

Osman and Piazzola [33] adapt the queueing Petri nets formalism to model asynchronous replication in a Cassandra data store. Cassandra is a column-based, scalable data store which offers following multi-master replication and data distribution [34]. The *key-value* space is mapped onto a ring. The ring is then split into ranges and each of the cluster members is assigned to one or more key subsets. Database clients can contact any of the cluster nodes, which then becomes a coordinator node. Each of the cluster nodes has the knowledge about the mapping between other nodes and data ranges. Cassandra allows to configure a *replication factor* (RF) variable which determines number of nodes that store replicated data. The replication process is asynchronous and thus offers lower response times. It may result, however, in the lack of data consistency. The trade-off between those two behaviors is controlled by the user-defined *consistency level* (CL), denoting the number of nodes in which a given operation must succeed before the coordinator node responds to the client.

Osman and Piazzola [33] point out the significant role of QPNs in capturing crucial aspects of asynchronous replication: scheduling, synchronization and blocking request in a particular node. The model was validated for read operation workload.

In the QPN (see Fig. 8), the database client is represented with a queueing place with exponential think time. The client sends *read* tokens and *enter-cluster* immediate transition fires. This results in random distribution of the incoming requests. Let i be the index of the node chosen to be the coordinator. A read token is placed in the *enter-node$_i$* – a timeless queueing place with the FIFO scheduling strategy. The number of parallel requests which may be processed by the node is controlled with thread tokens. The model was initialized with 32 tokens placed in *threads$_i$* ordinary place. When at least one thread token is in the threads$_i$ place, at cluster node i is modeled by the timed queueing place *node$_i$* with exponentially distributed service time, single server and processor sharing discipline. The mechanism caused by process-request$_i$ differs depending on whether node i stores data for the requested key or not.

Local and Non-Local Requests Processing. Since each of the nodes stores only a subset of data, coordinator node$_i$ might not hold data for the requested key. The QPN model described differentiates processing local and non-local requests. For local operations, an immediate transition process-request$_i$ relocates the read token from the enter-node$_i$ place to the node$_i$ timed queueing place. It is noted that the remote request does not affect the nodes' ability to process local operations, as it does not use thread tokens. Coordinator node i routes the request and blocks it until it receives responses from CL other nodes. This is done by firing the immediate transition distribute-request$_i$. As a result, one read token is placed in *blocked$_i$* ordinary place and each of CL contacted servers receives *read$_i$*. Let j be the index of one of the contacted members, the token of color read$_i$ is put in its enter-node$_j$. The replica nodes contacted are chosen according to the consistency level and replication factor.

When read$_i$ token leaves the *node$_j$* place, the *distribute-request$_j$* immediate transition returns the thread token to *threads$_j$* place and deposits response token in the *replica-response$_i$* place of *node$_i$*. When the number of tokens in the replica-response$_i$ place equals CL, the immediate transition unblock$_i$ fires. One read token is passed from the blocked$_i$ place to the the exit-cluster place and CL tokens are removed from the replica-response$_i$ place. The read request is then transferred back to the client.

In contrast to remote requests, local requests address keys that are stored by node$_i$. When the immediate transition process-request$_i$ fires, it removes both the read token from the enter-node$_i$ place and the thread token from the threads$_i$ place. Additionally, it deposits a *local-read* token in the timed queueing place node$_i$. In order to synchronize replies, a read token is placed in blocked$_i$. After node i processes the local request, it fires the distribute-request$_i$ transition. Thread token is returned and one token is placed in replica-response$_i$. If $\mathrm{CL} > 1$, the distribute-request$_i$ transition will deposit a read$_i$ token in the *enter-node* places of $\mathrm{CL} - 1$ other nodes and the request is processed as a remote request. When the number of tokens in the replica-response$_i$ place reaches a consistency level, unblock$_i$ transition fires. One read token is replaced from the blocked$_i$ place to the exit-cluster place and CL tokens are removed from the replica-response$_i$ place. The read request is then transferred back to the client.

4.3. MongoDB Replication Modeling

The model described can be adapted to evaluate the performance of other NoSQL data stores which have implemented asynchronous replication. Although MongoDB [35] is a document-oriented database and uses different high availability and scalability mechanisms, the request response time may be obtained with an analogical QPN. To avoid a single point of failure and to ensure the balancing of load among data clusters, MongoDB has developed replica sets [35]. Replica set is a group of mongod processes which hold the same set of data. In the group, one of the nodes acts as a master node and receives all write operations. Such a node is called the primary replica set member. Secondary replica set members are slave nodes to which all write operations are asynchronously replicated from the primary member. Read operations may be routed

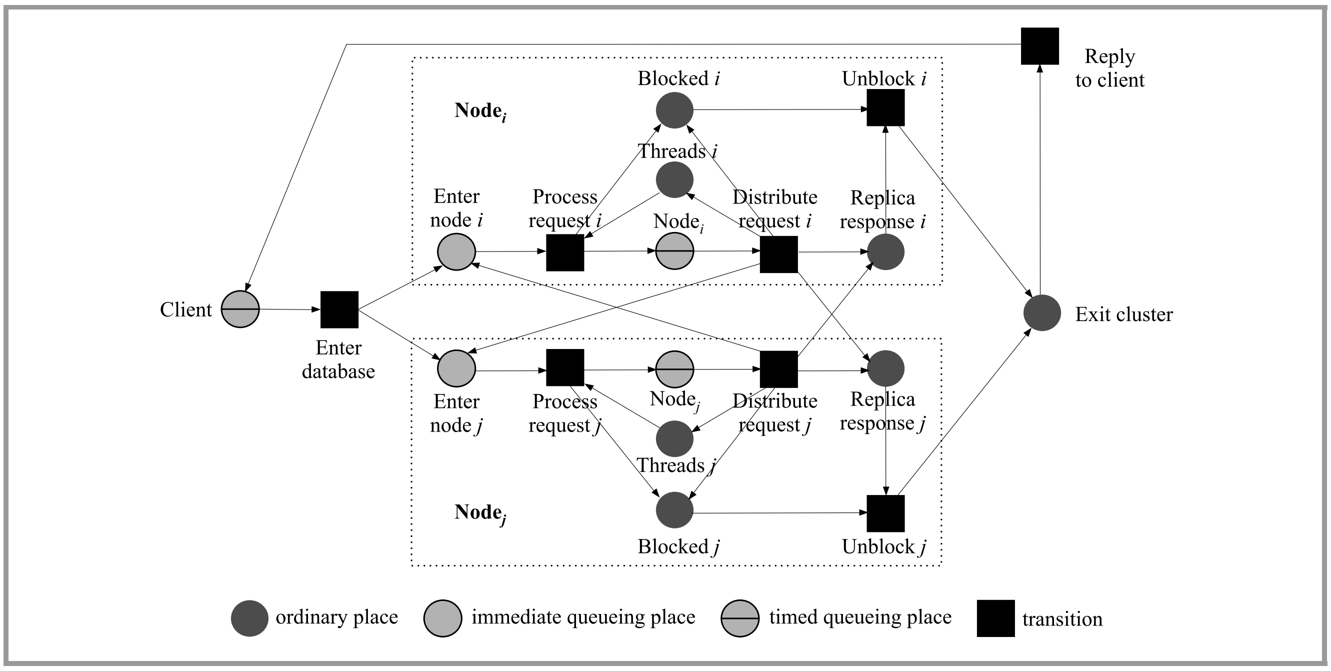

Fig. 8. QPN model of replication in a Cassandra cluster.

both to primary or secondary nodes, depending on the user-defined read preference variable. In the case of failure of the primary node, one of the running servers is chosen to be the new primary node. To allow horizontal scalability, MongoDB introduces *sharding* [35]. Sharding is automated data partitioning at the collection level which is based on the arbitrary shard key values. A shard key has to be an immutable field which exists in every document in the sharded collection. A shard is a subset of the collection data and is advised to be deployed as a replica set. Another component of a sharded cluster is the config database, storing meta data of the cluster. In particular, it contains mapping between data subsets and shards. A client sends a query to the router process *mongos*, which contacts the config server to retrieve information necessary for correct routing of the query to the shards. Depending on the requested shard key values, queries might be performed in one shard or distributed to several shards.

Applying the Osman and Piazzola [33] model for a replica set of a non-sharded database is straightforward. The number of the nodes holding the data in the replica set is equal to the replication factor RF. Similarly to the Cassandra data store, MongoDB allows the user to configure consistency level CL by setting read and write concerns. Depending on the read preference, the primary node or one of the secondary nodes receives the read request. Then, the read operation is processed similarly to a local request in the coordinator node, as described in the previous model.

The QPN formalism described sheds a light on the way future sharded cluster models may be constructed. Once a request arrives to the mongos process, it might be evaluated in a similar manner as remote requests processed by the coordinator node. However, communication between the router and the config database should be incorporated. Based on the knowledge from the config server, the router distributes the request to one or more shards. The number of target shards plays a role that is analogical to the consistency level of the previous model.

The above consideration may lead to the construction of a QPN model of replication and data distribution in the MongoDB cluster. However, such a work requires additional research. The complexity of QPN for a cluster composed of shards deployed as replica sets should be examined.

4.4. Database Contention Performance Modeling

Coulden *et al.* [36] construct a QPN model of a table-level database concurrency control through a strict two-phase locking protocol (Strict 2PL). Write operations hold exclusive locks on data objects, while read requests hold

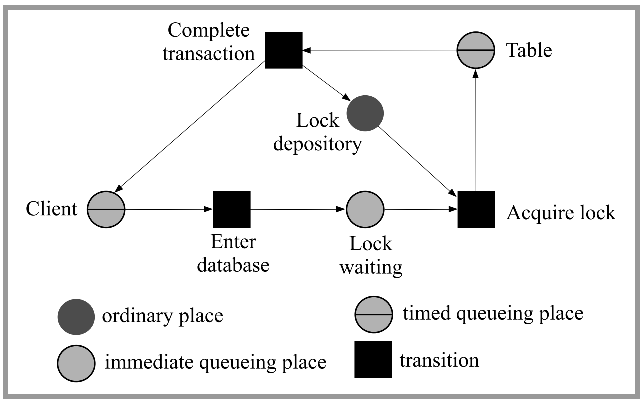

Fig. 9. QPN model of table level locking.

shared locks. A shared lock on a given data object may be acquired only when the object is not covered with an exclusive lock. An exclusive lock may be acquired on an object only when it is not covered with any other lock. This phenomenon can be captured with an ordinary repository place containing lock tokens. A shared transaction uses one lock token, while exclusive request requires the maximum number of tokens in the system. The database client is represented as a timed queueing place. The table accessed is represented as a timed $G/M/\infty$ queueing place with an infinite server queue. Figure 9 presents QPN of a table-level contention. The model was validated with the use of the *PostgreSQL* database management system [36].

5. Conclusion

In this paper a review of existing database performance models developed within the queueing theory approach was presented. The vast majority of conducted studies investigate the performance of relational databases. Only one of the models presented deals with the replication mechanism in the NoSQL Cassandra data store. The above-mentioned approach can be used to model performance of other replicated and distributed non-relational data stores, such as MongoDB. We claim that future studies should be pursued to rearrange the presented methodologies in order to evaluate the performance of NoSQL data stores. Moreover, while extensive research focuses on particular database components or constructs, the number of methodologies capable of mapping database system specifications onto queueing network models continues to be low. We believe that such studies should be pursued to develop methodologies for industrial applications.

Acknowledgements

This research was supported by the Polish National Science Centre (NCN) under the grant no. 2015/17/B/ST6/01885.

References

[1] M. Nicola and M. Jarke, "Performance modeling of distributed and replicated databases", *IEEE Trans. on Knowl. and Data Engin.*, vol. 12, pp. 645–672, no. 4, 2000 (doi: 10.1109/69.868912).

[2] R. Osman and W. J. Knottenbelt, "Database system performance evaluation models: A survey", *Perform. Eval.*, vol. 69, pp. 471–493, no. 10, 2012 (doi: 10.1016/j.peva.2012.05.006).

[3] L. Kleinrock, *Queueing Systems. Volume 1: Theory*. New York: Wiley-Interscience, 1975 (ISBN: 9780471491101).

[4] E. G. Coffmann, E. Gelenbe, and B. Plateau, "Optimization of the number of copies in a distributed data base", *IEEE Trans. on Software Engin.*, vol. SE-7, no. 1, pp. 78–84, 1981 (doi: 10.1109/TSE.1981.234510).

[5] F. Bacelli and E. Coffmann, "A database replication analysis using an M/M/m queue with service interruptions", in *Proc. of the 1982 ACM SIGMETRICS Conf. on Measur. and Model. of Comp. Syst. SIGMETRICS'82*, Seattle, VA, USA, 1982, pp. 102–107 (doi: 10.1145/1035332.1035309).

[6] R. Nelson and R. Iyer, "Analysis of a replicated data base", *Perform. Eval.*, vol. 5, no. 3, pp. 133–148, 1985 (doi: 10.1016/0166-5316(85)90008-2).

[7] E. Arzuaga and D. S. Kaeli, "An M/G/1 queue model for multiple applications on storage area networks", in *Proc. for the 11th Worksh. on Comp. Architec. Eval. using Commercial Workloads CAECW-11*, Salt Lake City, UT, USA, 2008, pp. 25–32.

[8] M. Dellkrantz, M. Kihl, and A. Robertsson, "Performance modeling and analysis of a database server with write-heavy workload", in *Proc. 1st Eur. Conf. on Serv.-Orient. and Cloud Comput. ESOCC 2012*, Bertinoro, Italy, 2012, pp. 184–191, 2012 (doi: 10.1007/978-3-642-33427-6_13).

[9] M. Kihl, P. Amani, A. Robertsson, G. Radu, M. Dellkrantz, and B. Aspernas, "Performance modeling of database servers in a telecommunication service management system", in *Proc. 7th Int. Conf. on Digit. Telecommun. ICDT 2012*, Chamonix, France, 2012, pp. 123–129.

[10] J. Mc Dermott and R. Mukkamala, "Performance analysis of transaction management algorithms for the SINTRA replicated architecture database systems", in *Proc. of the IFIP WG11.3 Working Conf. on Datab. Secur. VII*, Lake Guntersville, Alabama, USA pp. 215–234, 1993.

[11] B.-C. Jenq, B. C. Twichell, and T. W. Keller, "Locking performance in a shared nothing parallel database machine", *IEEE Trans. on Knowl. and Data Engin.*, vol. 1, no. 4, pp. 530–543, 1989 (doi: 10.1109/69.43427).

[12] H. Garcia-Molina, "Performance of the update algorithms for replicated data in a distributed database", Ph.D. Thesis, Stanford University, Stanford, CA, USA, 1979 [Online]. Available: http://www.dtic.mil/dtic/tr/fulltext/u2/a075268.pdf

[13] H. Garcia-Molina and G. Wiederhold, "Read-only transactions in a distributed database", *ACM Trans. on Datab. Syst.*, vol. 7, no. 2, pp. 209–234, 1982 (doi: 10.1145/319702.319704).

[14] R. D. van der Mei, A. R. de Wilde, and S. Bhulai, "A method for approximating the variance of the sojourn times in star-shaped queueing networks", *Stochastic Models*, vol. 24, no. 3, pp. 487–501, 2008 (doi: 10.1080/15326340802232327).

[15] D. Liang and S. Tripathi, "Performance analysis of long-lived transaction processing systems with rollbacks and aborts," *IEEE Trans. on Knowl. and Data Engin.*, vol. 8, no. 5, pp. 802–815, 1996 (doi: 10.1109/69.542031).

[16] M. J. Carey and M. Liviny, "Distributed concurrency control performance: A study of algorithms, distribution, and replication", in *Proc. of the 14th Int. Conf. on Very Large Datab. VLDB'88*, Los Angeles, CA, USA, 1988, pp. 13–25.

[17] B. Ciciani, D. M. Dias, and P. S. Yu, "Analysis of replication in distributed database systems", *IEEE Trans. on Knowl. and Data Engin.*, vol. 2, no. 2, pp. 247–261, 1990 (doi: 10.1109/69.54723).

[18] B. Ciciani, D. Dias, M., and P. S. Yu, "Analysis of concurrency-coherency control protocols for distributed transaction processing systems with regional locality", *IEEE Trans. on Knowl. and Data Engin.*, vol. 18, no. 10, pp. 899–914, 1992 (doi: 10.1109/32.163606).

[19] S. Banerjee, V. O. Li, and C. Wang, "Performance analysis of the send-on-demand: A distributed database concurrency control protocol for high-speed networks", *Comp. Commun.*, vol. 17, no. 3, pp. 189–204, 1994 (doi: 10.1016/0140-3664(94)90005-1).

[20] S. Y. Hwang, K. Lee, and Y. H. Chin, "Data replication in a distributed system: A performance study", in *Proc. 7th Int. Conf. Database and Expert Systems Applications*, Zurich, Switzerland, 1996, pp. 708–717 (doi: 10.1007/BFb0034724).

[21] K. K. Leung, "An update algorithm for replicated signaling databases in wireless and advanced intelligent networks", *IEEE Trans. on Comp. – Special issue on mobile computing archive*, vol. 46, no. 3, pp. 362–367, 1997 (doi: 10.1109/12.580431).

[22] E. Born, "Analytical performance modelling of lock management in distributed systems", *Distributed Systems Engineering*, vol. 3, no. 1, pp. 68–76, 1996 (doi: 10.1088/0967-1846/3/1/008).

[23] D. A. Menascé and M. N. Bennani, "Analytic performance models for single class and multiple class multithreaded software servers", in *Proc. 32nd Int. Computer Measur. Group Conf.*, Reno, NV, USA, 2006, pp. 475–482.

[24] B. M. M. Gijsen, R. D. van der Mei, P. Engelberts, J. L. van den Berg, and K. M. C. van Wingerden, "Sojourn time approximations in queueing networks with feedback", *Perform. Eval.*, vol. 63, no. 8, pp. 743–758, 2006 (doi: 10.1016/j.peva.2005.08.002).

[25] A. Thomasian and K. Ryu, "A decomposition solution to the queueing network model of the centralized DBMS with static locking", in *Proc. of the Int. Conf. on Measur. and Model. of Comp. Syst. SIGMETRICS 1983*, Minneapolis, MN, USA, 1983, pp. 82–92, 1983 (doi: 10.1145/800040.801397).

[26] R. J. T. Morris and W. S. Wong, "Performance analysis of locking and optimistic concurrency control algorithms", *Perform. Eval.*, vol. 5, no. 2, pp. 105–118, 1985 (doi: 10.1016/0166-5316(85)90043-4).

[27] P. S. Yu, S. Balsamo, and Y. Lee, "Dynamic transaction routing in distributed database systems", *IEEE Trans. on Software Engin.*, vol. 14, no. 9, pp. 1307–1318, 1988 (doi: 10.1109/32.6174).

[28] N. Tomov *et al.*, "Analytical response time estimation in parallel relation database systems", *Parallel Computing*, vol. 30, no. 2, pp. 249–283, 2004 (doi: 10.1016/j.parco.2003.11.003).

[29] R. Osman, I. Awan, and M. E. Woodward, "QuePED: Revisiting queueing networks for the performance evaluation of database designs", *Simulation Modelling Practice and Theory*, vol. 19, no. 1, pp. 251–270, 2011 (doi: 10.1016/j.simpat.2010.06.010).

[30] R. Ramakrishnan and J. Gehrke, *Database Management Systems*. Boston: McGraw-Hill, 2002 (ISBN: 9780072465631).

[31] F. Bause, "Queueing Petri nets – a formalism for the combined qualitative and quantitative analysis of systems", in *Proc. of 5th Int. Worksh on Petri Nets and Perf. Models*, Toulouse, France, 1993 (doi: 10.1109/PNPM.1993.393439).

[32] S. Kounev, S. Spinner, and P. Meier, "Introduction to queueing Petri nets: Modeling formalism, tool support and case studies", in *Proc. of the 3rd ACM/SPEC Int. Conf. on Perform. Engin. ICPE 2012*, Boston, MA, USA, 2012, pp. 9–18 (doi: 10.1145/2188286.2188290).

[33] R. Osman and P. Piazzolla, "Modeling replication in noSQL datastore", in *Proc. 11th Int. Conf. Quantitat. Eval. of Syst. QEST 2014*, Florence, Italy, 2014, pp. 194–209 (doi: 10.1007/978-3-319-10696-0_16).

[34] R. Cattell, "Scalable SQL and noSQL data stores", *ACM SIGMOD Record*, vol. 39, no. 4, pp. 12–27, 2010 (doi: 10.1145/1978915.1978919).

[35] K. Chodorov, *MongoDB: The Definitive Guide*, 2nd ed. Sebastopol, CA, USA: O'Reilly Media, Inc., 2013 (ISBN 9781449344795).

[36] D. Coulden, R. Osman, and W. J. Knottenbelt, "Performance modelling of database contention using queueing Petri nets", in *Proc. of the 4th ACM/SPEC Int. Conf. on Perform. Engin. ICPE 2018*, Prague, Czech Republic, 2013 (doi: 10.1145/2479871.2479919).

Antonina Krajewska is a second-year Ph.D. student at the Systems Research Institute of the Polish Academy of Sciences and a fellow working on a research project titled "An energy-aware computer system for HPC computing" at the Faculty of Electronics and Information Technology of the Warsaw University of Technology. She has been with the Research and Academic Computer Network (NASK) since 2016. She holds an M.Sc. in Mathematics and an M.Sc. in Psychology from Warsaw University. Her research areas focus on modeling of performance of database systems, Markov chains and NoSQL data stores.
https://orcid.org/0000-0001-6626-5667
E-mail: antonina.krajewska@nask.pl
Research and Academic Computer Network (NASK)
Kolska 12
01-045 Warsaw, Poland