Postprint: NWR Database Write Latency Model Based on (n,r,k) Fork-Join Queue Analysis

Authors: Wang Huajin, Li Jianhui, SHEN Zhihong

Date: 2018-05-20T00:00:00+00:00

Abstract

Write latency estimation for NWR databases can be used to discover configuration combinations of node counts and replica factors that minimize cluster construction and operational costs. Existing approaches based on benchmarking or simulated queuing are constrained by specific test configurations and environments, yielding only coarse-grained results of how write latency varies with configuration changes. By analyzing the (n, r, k) Fork-Join queue structure of write operations in the NWR database Cassandra, we derive an analytical solution for the expected sojourn time of this class of queues and a theoretical model for NWR database write latency, which enables more comprehensive write latency analysis. The accuracy of the (n, r, k) queue analytical solution and the write latency model is validated on both simulated queues and Cassandra clusters.

Full Text

Preamble

Title: Write Latency Model for NWR Databases Based on (n, r, k) Fork-Join Queueing Analysis

Authors: Wang Huajin[1],[2], Li Jianhui[1]†, Shen Zhihong[1]
[1] Computer Network Information Center, Chinese Academy of Sciences, Beijing 100190, China
[2] University of Chinese Academy of Sciences, Beijing 100049, China

Abstract: Accurate estimation of write latency for NWR databases can inform the construction and operation of database clusters by identifying optimal combinations of cluster size and replication factor that minimize costs. Existing approaches based on benchmarking or queue simulation are limited to specific test configurations and environments, yielding only coarse-grained results on how latency varies with configuration. This paper presents the first

closed-form analysis of the (n, r, k) Fork-Join queueing process for write operations in Cassandra (a typical NWR database), deriving an analytical solution for the expected sojourn time of such queues and establishing a theoretical write latency model for NWR databases that enables more comprehensive latency predictions. The analytical solution for (n, r, k) queues and the write latency model are validated through experiments on both simulated queues and a Cassandra cluster.

Keywords: database queueing; replication factor; consistency level; write latency

0 Introduction

NWR is a popular consistency strategy for key-value databases: (1) each key-value pair is stored across N replicas on different nodes; (2) read operations must access at least R replicas to guarantee reading the latest data; (3) write operations must successfully write to at least W replicas; (4) $R + W > N$ ensures consistency. Amazon Dynamo[1] and Apache Cassandra[2] are two widely-used key-value databases employing the NWR strategy (hereinafter referred to as NWR databases).

In NWR databases, write operations are distributed to replicas via a Fork-Join queue: (a) a client write request is replicated (Forked) into N subtasks at the coordinator node, each dispatched to one of the N replica nodes storing the target record; (b) upon completing its write, each replica node sends an acknowledgment back to the coordinator; (c) after receiving (Joining) the W-th acknowledgment, the coordinator responds to the client that the write has succeeded. Clearly, write latency in NWR databases depends on the cluster size L, replication factor N, and consistency level W. Generally, larger L, larger N, and smaller W yield lower latency. However, larger L increases cluster construction costs, while larger N increases storage and operational costs. Since NWR database clusters are typically built to meet SLA (Service Level Agreement) requirements for write latency under specific target workloads, the common practice is to provision large-scale clusters with high replication factors to ensure latency requirements are met even under extreme load, often leading to wasted resources and energy. Accurate estimation of write latency across various consistency levels and load pressures can identify the minimal L and N combinations that satisfy latency requirements, providing crucial guidance for cost-minimized cluster construction and configuration.

Existing latency analysis methods for Cassandra and other NWR databases include: (a) Benchmarking-based approaches[3] , which use mainstream distributed database benchmarking tools to measure read/write latencies across various node counts, replication factors, consistency levels, and load pressures. However, since benchmarking cannot exhaustively test all configuration combinations, these works only provide coarse trends of latency variation with partial configurations, without proving universal applicability (completeness) of their

conclusions. (b) Simulation-based approaches [13], which construct queueing networks equivalent to database read/write execution processes and obtain latency data by running simulations rather than actual clusters, reducing testing overhead. Since simulation structures and parameters only approximate real clusters under specific configurations and loads, their results are less reliable than benchmarks, and conclusions remain coarse and incomplete.

This paper presents a theoretical write latency model for NWR databases. The model takes as input the expected write latency under the highest consistency level (ALL) across different configurations, and outputs estimated expected latencies for weaker consistency levels (ONE, QUORUM, etc.) under related configurations. Input parameters can be obtained either theoretically or via benchmarking. Using this model, equally intensive benchmarking yields more detailed and comprehensive latency conclusions. The theoretical derivation builds upon quantitative analysis of the Fork-Join queueing process in Cassandra writes. Existing research on queue sojourn times only provides coarse bounds[1] . This paper proves that the expected sojourn time of (n, r, k) queues is a linear combination of basic Fork-Join (r, r, k) queue sojourn times, transforming the difficult (n, r, k) queue analysis into a more tractable (r, r, k) queue analysis. Finally, by dissecting Cassandra' s write process queueing structure, we apply (r, r, k) queue analysis to practical database write latency estimation.

The main contributions are: (a) We derive the first exact analytical expression for the expected sojourn time of (n, r, k) Fork-Join queues, validated on simulated queues with independent and identically distributed (i.i.d.) exponential service times; (b) We propose the first theoretical write latency model for NWR databases, validated on an actual Cassandra cluster.

1 Non-Clearing (n, r, k) Fork-Join Queue Analysis

Consider a homogeneous cluster of n nodes, where each node processes subtasks with i.i.d. service times. Each node has a first-come-first-served subqueue with unlimited capacity. Let denote the job arrival rate to the cluster, the service rate of a subqueue (subtasks completed per unit time), and = / the load factor.

Definition 1 (Non-Clearing Fork-Join Queue). As shown in [Figure 1: see original paper], in a non-clearing Fork-Join queue with n subqueues: (1) each arriving job is replicated (Forked) into r subtasks of equal cost, distributed to r distinct nodes with equal selection probability; (2) upon completing a subtask, a node sends an acknowledgment back to the coordinator; (3) after receiving (Joining) the first k acknowledgments, the coordinator responds to the client that the job is complete; (4) remaining subtasks continue execution (non-clearing) but their results need not be processed by the coordinator.

Definition 2 (Expected Sojourn Time). The expected sojourn time T_{(n,r,k)}( ) of a Fork-Join queue is defined as: when the queue reaches

steady state under load factor , the expected time interval from when a new job is Forked at the coordinator until it completes Join at the coordinator.

1.1 Analysis of General Service Time Queues

Lemma. Consider two Fork-Join queues with identical service time distributions: one is a non-clearing (n, r, k) queue under load factor , the other is a non-clearing (r, r, k) queue under load factor (n/r) . Ignoring asymptotic independence[1] , these two queues have the same expected sojourn time: T_{(n,r,k)}( ) = T_{(r,r,k)}((n/r)).

Proof. In the (n, r, k) queue, each arriving Fork-Join job has r subtasks assigned to r distinct subqueues, waiting for the first k subtasks to complete. Since each node is selected with equal probability, each subqueue in the (n, r, k) queue experiences subtask arrival rate r/n, identical to the (r, r, k) queue' s subtask arrival rate . With identical service time distributions across all subqueues, all subqueues have the same expected length in steady state. Let random variable X_i denote the sojourn time of a subtask in subqueue i, and S_{i(k)} the k-th order statistic of order statistics S_{i(1)},⋯,S_{i(r)}. Ignoring asymptotic independence[1] :

T_{(n,r,k)}( ) = E[min_{1 i <⋯<i_k n} max{X_{i },⋯,X_{i_k}}] = E[S_{(k)}] = T_{(r,r,k)}((n/r))

Theorem 1 (Expected Sojourn Time of Non-Clearing (n, r, k) Fork-Join Queue). The expected sojourn time is:

T_{(n,r,k)}( ) = *{i=k}^r W_i · T*{(i,i,i)}(_i)

where the constant coefficients are:

W_i = A_i^k · (r choose i) · (n/r)^i · (1 - n/r)^{r-i}, for k i r

A_i^k = _{j=k}^i (i choose j) · (-1)^{i-j}

Proof. By the Lemma, T_{(n,r,k)}( ) = T_{(r,r,k)}((n/r)). From Theorem 5 in [16]:

T_{(r,r,k)}((n/r)) = *{i=k}^r A_i^k · (r choose i) · T*{(i,i,i)}((n/r))

Since T_{(i,i,i)}((n/r)) = (n/r)^i · T_{(i,i,i)}( ), the result follows.

Notably, Theorem 1 does not require subqueue service times to be independent.

Theorem 2 (Bounds for Non-Clearing (n, r, k) Fork-Join Queue). The expected sojourn time satisfies:

Lower bound: T_{(n,r,k)}( ) max{T_{Low}((n/r)), T_{Nelson}((n/r))}

Upper bound: T_{(n,r,k)}( ) T_{Up}((n/r))

where T_{Low}, T_{Nelson}, T_{Up} are defined in the Appendix.

Proof. For the clearing Fork-Join queue (where remaining tasks are immediately cleared after job completion), Theorem 4 in [14] gives a lower bound T_{Low}((n/r)). Theorem 10 in [16] provides a tighter lower bound T_{Nelson}((n/r)) for i.i.d. exponential service times. Since the non-clearing queue' s expected sojourn time is no less than the clearing queue' s, the lower bound holds. The upper bound follows from substituting the expectation of the k-th order statistic of r i.i.d. service times into Theorem 2.

Based on Theorems 1 and 2, computing T_{(n,r,k)}( ) reduces to computing the more tractable T_{(i,i,i)}( ) and the expected k-th order statistic of service times.

1.2 Estimation for Exponential Service Time Queues

Since accurate Nelson estimation methods exist for i.i.d. exponential Fork-Join queues[1] , we provide estimated sojourn times for this case in Theorem 3.

Theorem 3 (Expected Sojourn Time for Non-Clearing Fork-Join Queue with i.i.d. Exponential Service Times). The expected sojourn time is bounded by:

T_{(n,r,k)}^{Nelson}( ) = (1/) ·[*{i=1}^r (1/(i - (n/r)))* - {i=1}^{r-k} (1/(i + k - (n/r)))]

T_{(n,r,k)}^{Low}( ) = (1/) ·[H_r - H_{r-k} + (k/(r - (n/r)))]

T_{(n,r,k)}^{Up}( ) = (1/) ·[H_r - H_{r-k} + (k/(r - (n/r))) + (k(k-1)/(2(r - (n/r))^2))]

where H_m = _{i=1}^m (1/i) is the harmonic number.

Proof. Substituting Equation (6) from [17] into Theorem 1 yields T_{(n,r,k)}^{Nelson}( ). Substituting into Theorem 2' s lower bound gives T_{(n,r,k)}^{Low}( ). The upper bound follows from the expectation of the k-th order statistic of r i.i.d. exponential variables: E[X_{(k)}] = (1/) · (H_r - H_{r-k}). Using T_{(n,r,k)}^{Nelson} alone may introduce significant error[1] , so bounds are used to constrain the estimate.

[Figure 2: see original paper] compares simulated values (SIM) against estimates (APP) from Theorem 3 for i.i.d. exponential service times. For each (n,r,k) configuration, the simulator Forkulator-p[1] was subjected to job arrival streams at specified loads. After reaching steady state, 100,000 sample jobs were collected at 10% sampling rate. Error rate is calculated as |APP - SIM|/SIM.

The results show estimation errors are generally controlled, becoming significant (>25%) only under heavy load (> 0.8) when 2k < r. This occurs because Nelson's method has larger errors at high , while small k accumulates more error terms. Future work will improve estimation methods for exponential queues to reduce this error.

2 Cassandra Write Operation Queueing Analysis

Cassandra' s write execution architecture is shown in [Figure 3: see original paper]. The client submits write requests to a cluster of n nodes, with the connected node becoming the Coordinator. The Coordinator sends write commands to r nodes storing replicas (replication factor r). When acknowledgments from replica nodes reach k (the consistency level threshold), the Coordinator responds to the client. If the Coordinator itself is a replica node, it' s a local Coordinator; otherwise, it' s remote. By default, a key-value record is stored contiguously on r nodes clockwise from a position determined by hashing the key, ensuring balanced storage distribution.

Cassandra employs SEDA (Staged Event-Driven Architecture)[1] , expressing write operations as a serial sequence of steps: Native-Transport-Requests, Mutation, RequestResponse, etc. Each step contains a thread pool where idle threads fetch requests from an arrival queue. The write process: (a) client requests arrive at the Coordinator' s Native-Transport-Requests queue, which identifies replica nodes and transmits writes via MessagingService to each replica' s Incoming queue; (b) the Mutation stage at replicas fetches and executes writes (appending to commitlog and writing to memtable); (c) replicas send results back to the Coordinator' s Incoming queue; (d) the RequestResponse stage counts acknowledgment messages; (e) when the count reaches consistency level k, the final response is sent to the client. With unbounded queues and no timeout limits, all arrival queues can be approximated as non-lossy.

This execution process maps naturally to our non-clearing Fork-Join queue model. Treating each internal step as part of the service process yields the simplified Cassandra write queueing model in [Figure 4: see original paper].

Theorem 4 (Expected Write Latency Model for Cassandra on Homogeneous Clusters). The expected write latency is:

L_{(n,r,k,l)}( ) = l ⋅ L_{(1,1,1)}( ) + (1 - l) ⋅ R_{(n-1,r,k)}( )

where l is the fraction of local writes, L_{(1,1,1)}( ) is the expected latency for local writes (node count = replication factor = consistency level = 1), and R_{(n-1,r,k)}( ) is the expected latency for remote writes (excluding the Coordinator node).

Model Parameters: (a) l: local write fraction; (b) L_{(1,1,1)}( ): baseline latency; (c) R_{(n-1,r,k)}( ): remote write latency for i replicas. Total parameters: r - k + 4.

Parameter l can be derived from load balancing policies (e.g., l = 1 for local-only, l = 0 for remote-only, l = r/n for RoundRobin). L_{(1,1,1)}( ) and R_{(n-1,r,k)}( ) can be estimated from service time distributions (see Section 1.2 and general service time estimation methods[1] ,[2]) or measured via benchmarking.

3 Experiments

We measured write latencies on a 6-node homogeneous Cassandra cluster across replication factors (r = 1..5), consistency levels (k = 1, r/2 , r), load balancing policies, and load pressures ( ), comparing measured values against Theorem 4' s predictions. Measured values used Cassandra' s built-in tracing (microsecond precision, 6% sampling rate). The target throughput was controlled via exponentially-distributed inter-arrival times. Predictions substituted measured parameters into Theorem 4.

The testbed comprised 6 identical physical nodes: dual Intel Xeon E5-2603 v3, 64GB RAM, 3.6TB HDD, 10GbE interconnect, running Cassandra 3.11.0. Each replication factor had a dedicated keyspace with 100 million records.

3.1 Micro-Benchmarks

Micro-benchmarks repeatedly wrote to a single key-value record across configurations to: (1) obtain model parameters L_{(1,1,1)}( ) and R_{(n-1,r,k)}( ); (2) validate the model for single-write operations. Custom HostFilterPolicy implementations enforced all-local (l = 1) and all-remote (l = 0) writes. Sampling rate was 1% with 60,000 samples per configuration.

[Figure 5: see original paper] shows estimated vs. measured latencies. The model closely matches measurements, validating its applicability for single writes. Error rates are computed as |approx - bench|/bench. Note that represents target load; actual achieved load is lower due to Python script overhead and sleep() precision limitations.

3.2 Extended Benchmarks

Extended benchmarks executed mixed writes to different keys across configurations to: (a) obtain model parameters; (b) validate the model for mixed write workloads. The workload ensured uniform Coordinator selection and balanced replica request distribution across nodes. Tested policies: all-local, all-remote, and RoundRobin (l = r/n). Sampling rates: 1% for all-local/remote (4,200 samples), 6% for RoundRobin (60,000 samples).

[Figure 6: see original paper] shows results for mixed writes. Most estimates closely match measurements, validating the model' s applicability. Both measured and predicted latencies show the expected ordering: all-local < RoundRobin < all-remote, suggesting TokenAwarePolicy (similar to all-local) outperforms RoundRobin.

[Figure 7: see original paper] shows the measured remote write service time distribution (50,000 samples), which does not follow an exponential distribution. While estimation methods exist for general service time distributions[1] ,[2] , their accuracy depends on precise distribution measurement. The instability of our measured distributions highlights why theoretical Fork-Join queue research[1] ,[21]

is rarely validated on real clusters. Future work will develop more robust estimation methods for general service time distributions.

4 Discussion

As noted, the model' s inputs are expected latencies under the highest consistency level (ALL), with outputs being estimates for weaker consistency levels (ONE, QUORUM, etc.). While Section 3 validates the model, obtaining parameters via benchmarking is costly. An ideal approach would measure service time distributions under no load, then compute parameters for any load pressure. Future directions include: (a) improving upper bounds for clearing (n, r, k) queues using our non-clearing results; (b) incorporating SEDA thread pool sizes via LPS queueing theory[2] to reflect thread count effects; (c) more robust estimation for general service time distributions; (d) comprehensive workload models (mixed read/write ratios, data access patterns); (e) algorithms to compute cost-minimizing (L, N) configurations meeting SLA latency requirements.

5 Related Work

5.1 Fork-Join Queue Analysis

The (n, n, n) Fork-Join queue is the most basic form. For its expected sojourn time: Nelson et al.[1] provided approximations for exponential service times; Varma et al.[1] proposed interpolation methods; Thomasian et al.[2] used regression; Rizk et al.[22] gave martingale-based bounds; Fidler et al.[23] provided bounds for multi-stage networks. For clearing (n, k) queues: exact solutions exist only when n = k[2] ,[2] ; coarse bounds exist when n k[1] ,[21]. For non-clearing (n, r, k) queues: Fidler et al.[23] gave non-asymptotic statistical bounds; Wang et al.[1] provided exact solutions via linear transformations and improved clearing queue bounds. This paper extends Wang et al.[1] to give the first exact analytical solution for non-clearing (n, r, k) queues.

5.2 NWR Database Latency Analysis

Benchmarking-based studies[3] use YCSB[2] to analyze how node count, replication factor, consistency level, and load affect latency. Limited by incomplete configuration coverage, these works only provide coarse trends without proving generalizability. Simulation-based approaches [13] build equivalent queueing networks, determining parameters via benchmarking, then run simulations for target configurations. While reducing overhead, simulations imperfectly capture real cluster behavior, yielding less reliable and still incomplete results. In contrast, our theoretical model generalizes across all configuration variables, with benchmarking used only for parameter acquisition and validation, producing more detailed and complete conclusions.

6 Conclusion

Given the widespread use of NWR databases, analyzing performance impacts of node count, replication factor, consistency level, and load pressure is increasingly important. Existing benchmarking and simulation methods produce coarse, incomplete results. This paper: (a) derived the first exact analytical solution for the expected sojourn time of non-clearing (n, r, k) Fork-Join queues, validated on simulated queues with i.i.d. exponential service times; (b) proposed the first theoretical write latency model for NWR databases, reflecting impacts of node count, replication factor, consistency level, load pressure, and load balancing policies. The model inputs are expected latencies under ALL consistency (from theory or measurement), outputting estimates for weaker consistencies. Compared to prior work, our model yields more detailed and complete conclusions, validated on a real Cassandra cluster.

Future work includes: (a) improving clearing (n, r, k) queue bounds using our non-clearing results; (b) incorporating SEDA thread pool modeling; (c) robust estimation for general service times; (d) comprehensive workload modeling; (e) cost-minimization algorithms for cluster configuration.

References

[1] Decandia G, Hastorun D, Jampani M, et al. Dynamo [J]. ACM SIGOPS Operating Systems Review, 2007, 41(6): 205–220.

[2] Lakshman A, Malik P. Cassandra [J]. ACM SIGOPS Operating Systems Review, 2010, 44(2): 35.

[3] Dede E, Sendir B, Kuzlu P, et al. An evaluation of cassandra for hadoop [C]// Proc of IEEE International Conference on Cloud Computing. 2013: 494–501.

[4] Wang H, Li J, Zhang H, et al. Benchmarking replication and consistency strategies in cloud serving databases: hbase and cassandra [C]// Proc of Workshop on Big Data Benchmarks, Performance Optimization, and Emerging Hardware. 2014: 71–82.

[5] Kuhlenkamp J, Klems M, Röss O. Benchmarking scalability and elasticity of distributed database systems [C]// Proc of the VLDB Endowment. 2014: 1219–1230.

[6] Huang X, Wang J, Yu P S, et al. An experimental study on tuning the consistency of nosql systems [J]. Concurrency and Computation: Practice and Experience, 2017, 29(12): e4129.

[7] Huang X, Wang J, Bai J, et al. Inherent replica inconsistency in cassandra [C]// Proc of IEEE International Congress on Big Data. 2014: 740–747.

[8] Osman R, Piazzolla P. Modelling replication in nosql datastores [C]// Proc of International Conference on Quantitative Evaluation of Systems. 2014: 1–

16.

[9] Gandini A, Gribaudo M, Knottenbelt W J, et al. Performance evaluation of nosql databases [C]// Proc of European Workshop on Performance Engineering. 2014: 16-29.

[10] Pérez-Miguel C, Mendiburu A, Miguel-Alonso J. Modeling the availability of cassandra [J]. Journal of Parallel and Distributed Computing, 2015, 86: 45-58.

[11] Niemann R. Evaluating the performance and energy consumption of distributed data management systems [C]// Proc of the 10th International Conference on Global Software Engineering Workshops. 2015: 27-34.

[12] Dipietro S, Casale G, Serazzi G. A queueing network model for performance prediction of apache cassandra [C]// Proc of the 10th EAI International Conference on Performance Evaluation Methodologies and Tools. New York: ACM Press, 2017: 186-193.

[13] Huang X, Wang J, Qiao J, et al. Performance and replica consistency simulation for quorum-based nosql system cassandra [C]// Proc of the 50th Annual Allerton Conference on Communication, Control, and Computing. 2012: 326-333.

[14] Joshi G, Soljanin E, Wornell G. Efficient redundancy techniques for latency reduction in cloud systems [J]. ACM Trans on Modeling and Performance Evaluation of Computing Systems, 2017, 2(2): 12:1-12:30.

[15] Wang W, Harchol-Balter M, Jiang H, et al. Asymptotic response time analysis for multi-task parallel jobs [EB/OL]. (2017/10/27) [2017/11/12]. Arxiv: 1710.00296 [cs.PF].

[16] Wang H, Li J, Shen Z, et al. Approximations and bounds for (n, k) fork-join queues: a linear transformation approach [EB/OL]. (2017/09/08) [2017/09/12]. Arxiv: 1707.08860 [cs.PF].

[17] Nelson R, Tantawi A N. Approximate analysis of fork/join synchronization in parallel queues [J]. IEEE Trans on Computers, 1988, 37(6): 739-743.

[18] Welsh M, Culler D, Brewer E. Seda: an architecture for well-conditioned, scalable internet services [J]. ACM SIGOPS Operating Systems Review, 2001, 35(5): 230-243.

[19] Varma S, Makowski A M. Interpolation approximations for symmetric fork-join queues [J]. Performance Evaluation, 1994, 20(1-3): 245-265.

[20] Thomasian A, Tantawi A N. Approximate solutions for m/g/1 fork/join synchronization [C]// Proc of Winter Simulation Conference. San Diego: Society for Computer, 1994: 361-368.

[21] Joshi G, Liu Y, Soljanin E. Coding for fast content download [C]// Proc

of the 50th Annual Allerton Conference on Communication, Control, and Computing. 2012: 326-333.

[22] Rizk A, Poloczek F, Ciucu F. Computable bounds in fork-join queueing systems [C]// Proc of ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems. New York: ACM Press, 2015: 335-346.

[23] Fidler M, Jiang Y. Non-asymptotic delay bounds for (k, l) fork-join systems and multi-stage fork-join networks [C]// Proc of the 35th Annual IEEE International Conference on Computer Communications. 2016.

[24] Lee K, Pedarsani R, Ramchandran K. On scheduling redundant requests with cancellation overheads [J]. IEEE/ACM Trans on Networking, 2017, 25(2): 1279-1290.

[25] Gardner K, Zbarsky S, Doroudi S, et al. Reducing latency via redundant requests: exact analysis [C]// Proc of ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems. New York: ACM Press, 2015: 347-360.

[26] Cooper B F, Silberstein A, Tam E, et al. Benchmarking cloud serving systems with ycsb [C]// Proc of the 1st ACM Symposium on Cloud Computing. New York: ACM Press, 2010: 143-154.

[27] Towsley D, Rommel C G, Stankovic J A. Analysis of fork-join program response times on multiprocessors [J]. IEEE Trans on Parallel and Distributed Systems, 1990, 1(3): 286-303.

Note: Figure translations are in progress. See original paper for figures.