MADALGO
CENTER FOR MASSIVE DATA ALGORITHMICS


Darius Šidlauskas
Aarhus University


Christian S. Jensen
Aarhus University


Simonas Šaltenis
Aalborg University

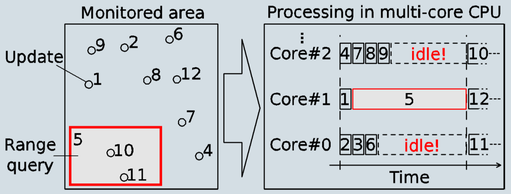
Danmarks Grundforskningsfond
Danish National Research Foundation

Processing of Extreme Moving-Object Update and Query Workloads in Main Memory

Introduction

Today, increased on-chip parallelism is a key means of improving processor performance. Moving-object workloads with long-running queries and massive numbers of updates render it particularly challenging to avoid inter-thread interference and thus achieve scalability.

Traditional database serializability requires extensive locking and implements *timeslice* semantics, meaning that a query reports precisely the objects in its range at a specific time instance. Consequently, operations are often blocked, leaving highly parallel CPUs underutilized, as processing cores remain idle.

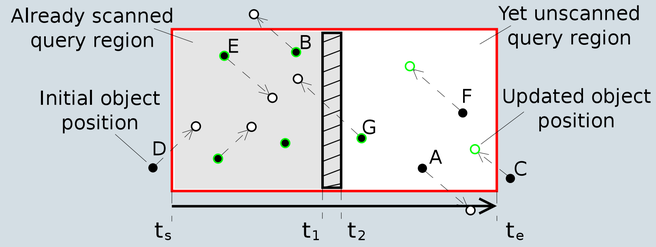


Query Semantics

Timeslice: $t_x < t_s$. All objects that are valid at a specific time instance (t_x) just before the query start time (t_s). A conventional serializable execution ensures such semantics.

Stale-timeslice: $t_x << t_s$. The time instance t_x is not guaranteed to be fresh. Snapshot-based processing provides such semantics [3].

Freshness: $t_x < t_s$ or $t_s < t_x < t_e$. As timeslice, but some objects can be *fresher*, i.e., updated during query processing from t_s to t_e. E.g., the green positions are reported (updates occur in $[t_1, t_2]$):

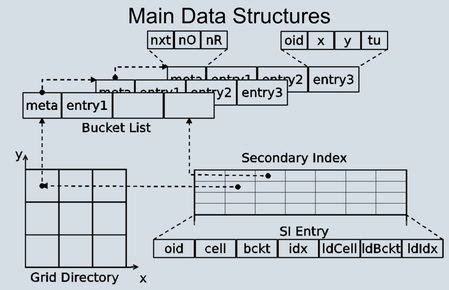


Key observation: $t_e - t_s < T_o$, where T_o is the time between two consecutive updates of an object.

Query staleness: the ratio of update s ignored by the query to the total number of objects. An update is ignored if it has a lower timestamp than a query, but is not taken into account by the query.

Implementations

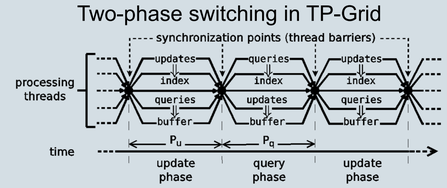
- Services queries via a uniform and static grid
- Services updates via a secondary index (bottom-up approach [1])
- Relies on hardware-assisted atomic operations for concurrency control
- Utilizes SIMD registers for atomic object data reads and writes



Semantics

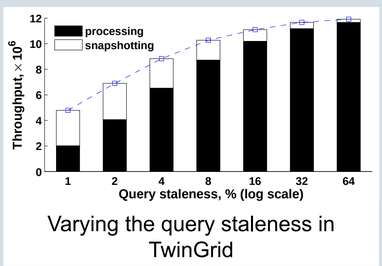
Query Results	Timeslice	Freshness
Up-to-date	u-Grid [2] Serial SerialPGrid	PGrid [4]
Stale/ delayed	TwinGrid [3] TP-Grid	----

Classification of considered approaches

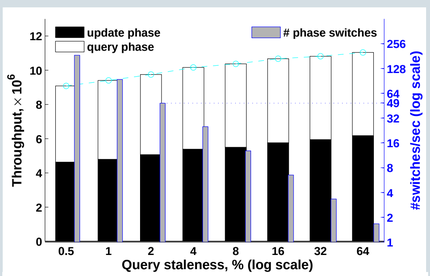


Empirical Study

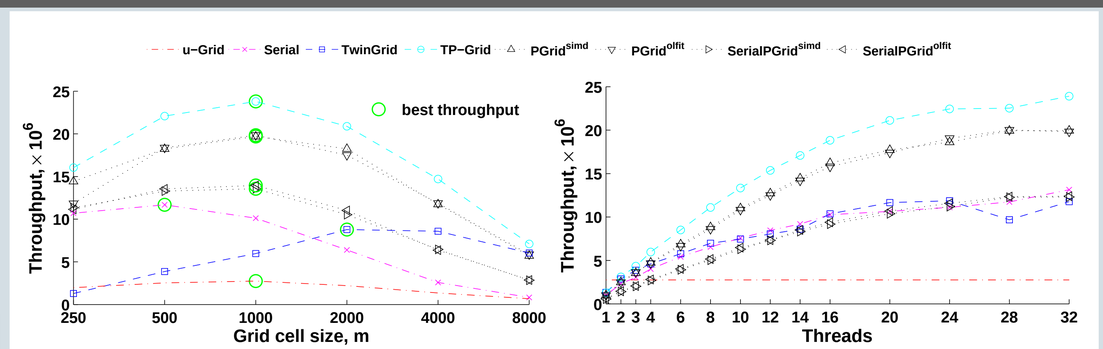
- Includes four diverse multi-core platforms
- Studies eight grid-based index variants
- Exercises the indexes under massive workloads containing tens of millions of moving vehicles simulated in a road network of Germany
- All figures show results obtained on a 16-core Intel E5-2670 machine with 32 hardware threads in total



Varying the query staleness in TwinGrid

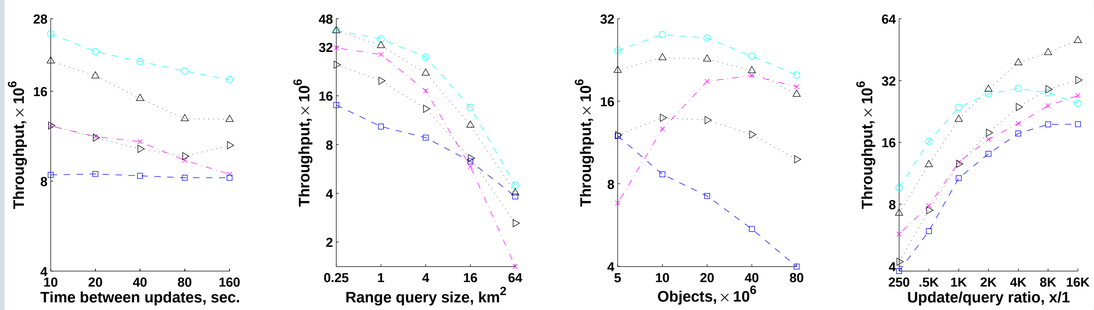


Varying the query staleness in TP-Grid



Optimal Grid Cell Size

Multi-threaded Scalability



Varying T_0 | Varying RQ Size | Varying #objects | Varying U/Q Ratio

Method	single-threaded read	single-threaded write	multi-threaded read	multi-threaded write
multi-read/write	4	4	-	-
pthread_mutex_t	70	70	517	521
pthread_rwlock_t	111	108	1150	1158
pthread_spinlock_t	25	25	153	170
1-byte latching using CAS	24	24	134	155
OLFIT	5	69	262	260
SIMD	3	3	19	18

Results from a micro-benchmark that measures CPU Cycles per 128-bit read/write using different synchronization methods.

References

[1] M. L. Lee et al. *Supporting frequent updates in R-trees: a bottom-up approach*. VLDB 2003.
[2] D. Šidlauskas et al. *Trees or grids? Indexing moving objects in main memory*. GIS 2009.
[3] D. Šidlauskas et al. *Thread-level parallel indexing of update intensive moving-object workloads*. SSTD 2011.
[4] D. Šidlauskas et al. *Parallel main-memory indexing for moving-object query and update workloads*. SIGMOD 2012.