FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Diplomarbeitspräsentation

Durability and Contention in Software Transactional Memory

Masterstudium:
Software Engineering & Internet Computing

Michael Schröder

Technische Universität Wien
Institut für Computersprachen
Arbeitsbereich: Programmiersprachen und Übersetzer
Betreuer: Univ.-Prof. Dr. Jens Knoop
Mitbetreuerin: Assoc. Prof. Gabriele Keller, UNSW

Software Transactional Memory

Software Transactional Memory (STM) vastly simplifies concurrent programming by grouping memory operations into atomic blocks.

The following Haskell function increments a transactional variable and returns its previous contents:

$$\begin{aligned} \textbf{inc}\ v = \ & \textbf{do}\ x \leftarrow \textbf{readTVar}\ v \\ & \qquad \textbf{writeTVar}\ v\ (x+1) \\ & \qquad \textbf{return}\ x \end{aligned}$$

To perform an STM computation and make its effects visible to the system, the function **atomically** :: **STM** $a \rightarrow$ **IO** a is used:

$$\textbf{atomically}\ (\textbf{inc}\ v)$$

Problem 1: Durability

Manipulating memory using STM is easy, but persisting those manipulations in a transactionally safe way is impossible.

Two obvious but unsafe ways of trying to add durability to STM:

1. $\textbf{do}\ x \leftarrow \textbf{atomically}\ m$
   $\quad \textbf{serialize}\ x$

Problem: serialization might fail after transaction committed

2. $\textbf{atomically}\ \$\ \textbf{do}\ x \leftarrow m$
   $\qquad \textbf{unsafeIOToSTM}\ (\textbf{serialize}\ x)$

Problem: transaction might abort or retry after serialization

STM Finalizers

Introduce a new STM primitive

$$\textbf{atomicallyWithIO} :: \textbf{STM}\ a \rightarrow (a \rightarrow \textbf{IO}\ b) \rightarrow \textbf{IO}\ b$$

which is like **atomically**, but additionally takes a *finalizer* — an I/O action that can depend on the result of the STM computation.

The finalizer is combined with the STM transaction such that:

1. The finalizer is only run if the transaction is guaranteed to commit.
2. The transaction only commits if the finalizer finishes successfully.

$$\frac{M;\ \Theta, \{\} \overset{*}{\Rightarrow} \texttt{return}\ N;\ \Theta', \Delta' \qquad F\ N;\ (\Theta \cup \Delta') \overset{*}{\rightarrow} \texttt{return}\ P;\ \hat{\Theta}}{\mathbb{P}[\texttt{atomicallyWithIO}\ M\ F];\ \Theta \rightarrow \mathbb{P}[\texttt{return}\ P];\ \Theta' \cup \hat{\Theta}}\ (\textit{ARET})$$

$$\frac{M;\ \Theta, \{\} \overset{*}{\Rightarrow} \texttt{return}\ N;\ \Theta', \Delta' \qquad F\ N;\ (\Theta \cup \Delta') \overset{*}{\rightarrow} \texttt{throw}\ P;\ \hat{\Theta}}{\mathbb{P}[\texttt{atomicallyWithIO}\ M\ F];\ \Theta \rightarrow \mathbb{P}[\texttt{return}\ P];\ \hat{\Theta}}\ (\textit{ATHROW2})$$

the design is formalized by an operational semantics

- durability is now trivially possible:

  $$\textbf{atomicallyWithIO}\ m\ \textbf{serialize}$$

- more generally, finalizers enable interactive transactions
- potential foundation for a distributed STM
- http://github.com/mcschroeder/ghc

Problem 2: Contention

Many standard data structures, when used in a transactional setting, cause unreasonably high numbers of conflicts.

Consider **TVar** (**HashMap** a):

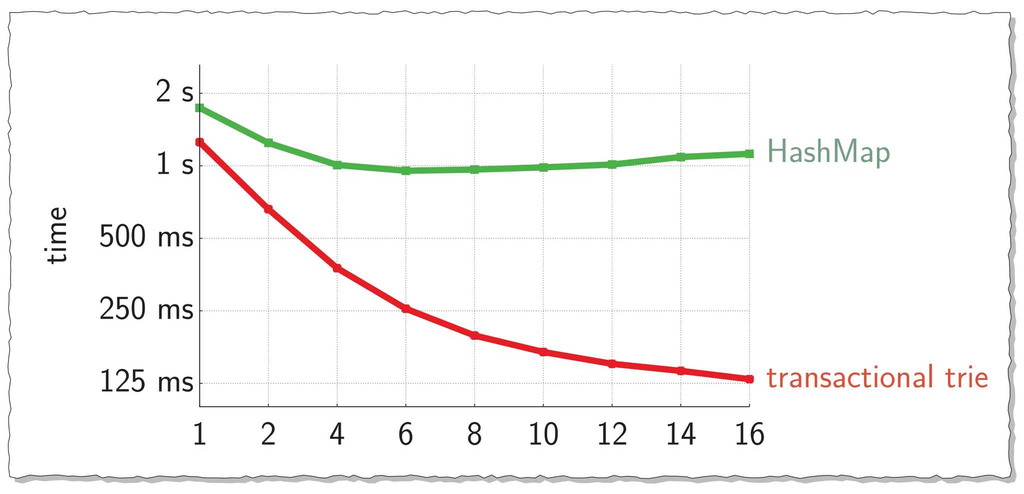
- any change to the container invalidates all other transactions
- but we should only care about the subset relevant to our transaction
- if transaction $\boldsymbol{A}$ updates element $\boldsymbol{k_1}$ and transaction $\boldsymbol{B}$ deletes element $\boldsymbol{k_2}$ and if $\boldsymbol{k_1} \neq \boldsymbol{k_2}$, then there should be no conflict

Transactional Trie

The **transactional trie** is a new contention-free data structure, specifically tailored to the needs of transactional concurrency.

- based on the lock-free concurrent trie
- uses **unsafeIOToSTM** to perform atomic compare-and-swap operations independently of the surrounding STM transaction
- to preserve safety, leaves are stored as **TVar** (**Maybe** a)
- once a leaf is added, it is never removed

empirical evaluation of 200 000 random transactions on an Amazon EC2 C3 extra-large instance with 16 cores



- eliminates all spurious conflicts
- up to 8 times faster and using almost 10 times less memory
- http://hackage.haskell.org/package/ttrie

An example application using both finalizers and transactional tries to build a lightweight database framework on top of STM is available at http://github.com/mcschroeder/social-example

Kontakt: michael.schroeder@alumni.tuwien.ac.at