Adapting to Point Contention with Long-Lived Adaptive Safe Agreement

Hagit Attiya  
Department of Computer Science  
Technion

Collecting Information in Asynchronous Shared-Memory Systems

Need to collect information in order to coordinate…
When only few processes participate, reading one by one is prohibitive …
Would like to have adaptive step complexity

Adaptive Step Complexity
A function of the number of active processes
Total contention: The number of processes that (ever) take a step during the execution

Collecting and store algorithms that are adaptive to total contention
[Attiya, Fouren & Gafni] … [Afek & De Levie]
- Renaming
- Atomic snapshots
Be More Adaptive?

- In a *long-lived* setting...

- ...processes come and go.

- What if many processes start the execution, then stop participating?

  - ...then start again...

  - ...then stop again...

Adapting to Point Contention

- The step complexity is a function only of the number of *currently* active processes.

  **Point contention of an operation:** Max number of processes taking steps *together* during its interval

A Weaker Notion: Interval Contention

- The step complexity is a function only of the number of *currently* active processes

  **Interval contention of an operation:** Max number of processes taking steps during its interval

  Always larger than the point contention

  ┌─────────┐ ┌─────────┐ ┌─────────┐ ┌─────────┐
  │          │ │          │ │          │ │          │
  └─────────┘ └─────────┘ └─────────┘ └─────────┘

Talk Outline

- What it means to be *dynamically* adaptive

- How to be adaptive?

  - The safe agreement object

  - An adaptive safe agreement object

  └ One-shot and long-lived └ Adaptive renaming └ Collecting information (adaptively)

- Extensions and connections
**Safe Agreement: Specification**

Separate the voting / negotiation on a decision from figuring the outcome.

Two wait-free procedures: Propose and Read.

Validity of non-Ø views

Agreement on non-Ø views returned by Read.

Termination: If all processes that invoked Propose return, then Read returns non-Ø view.

**Safe Agreement: Implementation**

Use an atomic snapshot object and an array R [Borowsky & Gafni].

Propose(info):
- update(info)
- scan
- write returned view to R[i]

Read() returns view
- find minimal view C written in R
- if all processes in C wrote their view
- return C
- else return Ø

**Safe Agreement: Safety**

Let C be the minimal view returned by any scan.

Can prove that all non-Ø views are equal to C.

**Safe Agreement: Liveness Properties**

Clearly, both procedures are wait-free.

- But Read may return a meaningless value, Ø.

If some process invokes Propose, then after all processes that invoke Propose return, a Read returns a non-Ø value.
Safe Agreement: Winners

Even better...

A Read by some process in C returns a non-Ø value. E.g., the last process in C to write its view. These processes are called winners.

Safe Agreement and the BG Simulation

Safe agreement was introduced by Borowsky & Gafni for fault-tolerant simulation of wait-free algorithms. Abstracted by Lynch&Rajsbaum.

- Different interface
  - Propose and Read not separated
  - No Ø response for read
  - Complicates the simulation

They also missed an interesting feature…

Safe Agreement: Concurrency

All processes in C execute Propose concurrently. In particular, all winners.

Use a doorway variable inside to avoid unnecessary update / scan.
Safe Agreement: Concurrency

All processes in C execute \textbf{Propose} concurrently.

In particular, all winners use a doorway variable \textit{inside} to avoid unnecessary update / scan.

Adaptive Safe Agreement

\begin{verbatim}
Propose( info )
if not inside then
  inside = true
  update( info )
  scan
  write the returned view
  return( true )
else return( false )
\end{verbatim}

Concurrency: If a process returns false then some "concurrent" process is accessing the object.

Long-Lived Adaptive Safe Agreement

Enhance the interface with a generation number (nondecreasing counter).

Validity, agreement and termination as before but relative to a single generation.

\textbf{Concurrency:} If a process returns false, c then some process is concurrently in generation c of the object.

Long-Lived Adaptive Safe Agreement

Synchronization: processes are inside the same generation simultaneously.

\begin{itemize}
  \item Their number \leq point contention
  \item Can employ algorithms adaptive to total contention within each generation
    \begin{itemize}
      \item e.g., atomic snapshots
    \end{itemize}
\end{itemize}
Many copies of one-shot safe agreement
- count points to the current copy
- Winners of each copy are synchronized
  - Increase count by 1.
  - Monotone

When all processes release a generation, open the next generation by enabling the next copy

Catching Processes with Safe Agreement
- When processes access an adaptive long-lived safe agreement object simultaneously, at least one wins
- If a process accesses an adaptive long-lived safe agreement object and does not win, some other process is accessing the object

Good for adaptivity...

Renaming
- A process has to acquire a unique new name
  - Later release it
- The range of new names must be as small as possible
  - Preferably adaptive: depending only on the number of active processes
  - Must be at least \( 2k - 1 \)

Renaming is a building block for adaptive algorithms
- First obtain names in an adaptive range
- Then apply an ordinary algorithm using these names

Renaming using Long-Lived Safe Agreement
- Place objects in a row...
- Agreement in each long-lived safe agreement object
  - Uniqueness of names
Renaming: Complexity

Concurrency for each long-lived safe agreement object
- An object is skipped only due to a concurrent process
- A process skips \( r \) objects
  - \( r \) is the interval contention
  - Range of names \( \approx r^2 \)

return (4, rank in C)

We promised point contention

- \( P_i \) skips because of \( P_j \)
- \( P_j \) skips because of \( P_k \)
- \( P_k \) skips because of ...

They all overlap

Proof is subtle since a process skips an object either due to a concurrent winner or due to a concurrent non-winner in C (which it can meet again later in the row)

- Use a potential-function proof to show that a process skips \( \leq 2k-1 \) objects
- \( k \) is the point contention
  - Name \( \approx k^2 \)
  - \( f(k) \) step complexity

Store using Long-Lived Safe Agreement

Place objects in a row...

A winner adds the values of this generation’s candidates to a register associated with the object Agreement in each object and the synchronization property imply that the register records all values of all candidates
- Across generations
- Compact for specific functions / purposes
Adaptive Collect?

Go over the associated registers and read…

What if pw and all other stores complete?

A collect running solo still has to reach the object in which pw has written its value!

Making the Collect Adaptive

Before completing the store, bubble-up information the top of the array

But how?
No CAS, waiting, or locks…

Wrap-Up

Long-lived adaptive safe agreement objects can be combined with bubble-up to obtain adaptive (to point contention) algorithms for:

- Gathering & collecting information
- Atomic snapshots
- Immediate snapshots
- (2k-1)-renaming (optimal)

Even More…

The algorithms can be made fully adaptive

- Step complexity depends on processes really participating, not just “signing in”
  - Especially relevant in renaming-based algorithms
- Can bound their memory requirements
- But the bounds are not adaptive…
Space: The Final Frontier

- Improve the step complexity of the algorithms and reduce their space complexity
  - Lots of improvement recently for total contention
  - E.g., using randomization
- Algorithms whose space complexity is truly adaptive to point contention?
  - Currently, number of registers used depends on total contention
  - Allocated vs. used registers

Other Aspects

- Using stronger primitives (CAS…)
  - Promising for adaptive space complexity
    [Afek, Dauber, Touitou]
    [Herlihy, Luchangco, Moir]
- More modularity…
  - We made some progress with the long-lived adaptive safe agreement object
  - What about bubble-up?

Lower Bounds, Anyone?

- Non-constant number of multi-writer registers is needed for adaptive weak test&set
  [Afek, Boxer, Touitou]
  - Holds also for renaming and long-lived collect
- Non-constant number of multi-writer registers is needed for adaptive generalized weak test&set
  [Aguilera, Englert, Gafni]
  - Holds also for one-shot collect
- Linear number of multi-writer registers is needed for adaptive and efficient one-shot collect
  [Attiya, Fich, Kaplan]

At a Broader Perspective

Connections with recent research trends:
- Obstruction-free algorithms
  - Adapting to step contention
    [Attiya et al. DISC 2005], [Attiya et al. PODC 2006]
- Abortable / failing objects
- Population-oblivious algorithms