A F(ASTER) SET LIBRARY

FSET

- London Clojurians Meetup - Tue Jul 6th - 2021

AGENDA

- A quick introduction to set functions
- Current fset offering and quick demo
- How to approach performance work
- Trying fset in the wild
- Conclusions

SET ALGEBRA

```
(require '[clojure.set :as sets])

(sets/union #{1 2 3} #{4 2 6}) ;; #{1 4 6 3 2}

(sets/difference #{1 2 3} #{4 2 6}) ;; #{1 3}

(sets/intersection #{1 2 3} #{4 2 6}) ;; #{2}

(sets/subset? #{1 2} #{1 2 3}) ;; true

(sets/superset? #{:a :b :c} #{:a :c}) ;; true
```

RELATIONAL ALGEBRA

```
(def users
  #{{:user-id 1 :name "john"    :age 22 :type "personal"}
    {:user-id 2 :name "jake"    :age 28 :type "company"}
    {:user-id 3 :name "amanda" :age 63 :type "personal"}})

(def accounts
  #{{:acc-id 1 :user-id 1 :amount 300.45 :type "saving"}
    {:acc-id 2 :user-id 2 :amount 1200.0 :type "saving"}
    {:acc-id 3 :user-id 1 :amount 850.1 :type “debit"}})

;; SELECT users.user-id, accounts.acc-id,
;;        users.type as type, accounts,type as atype
;; FROM users
;; INNER JOIN accounts ON users.user-id = accounts.user-id;

(sets/project
  (sets/join users (sets/rename accounts {:type :atype}))
  [:user-id :acc-id :type :atype])

;; #{{:user-id 1, :acc-id 1, :type "personal", :atype "saving"}
;;   {:user-id 2, :acc-id 2, :type "company", :atype "saving"}
;;   {:user-id 1, :acc-id 3, :type "personal", :atype "debit"}}
```

OTHER HELPERS

```clojure
(require '[clojure.set :refer [rename-keys map-invert]])

(rename-keys {:a 1 :b 2 :c 3} {:a "AA" :b "B1" :c "X"})
;; {"AA" 1, "B1" 2, "X" 3}

(map-invert {:a 1 :b 2})
;; {1 :a, 2 :b}

(select-keys {:a 1 :b 2 :c 3} [:a :c])
;; {:a 1, :c 3}
```

SHOWCASE

- From clojure.set :
  - algebra: union (46%), difference (43%), intersection (43%), subset? (80%), superset? (80%)
  - relations: index (34%), join (70%), project (50%), select (25%)
  - other: map-invert, rename, rename-keys (40%)
- Also in fset: select-keys (55%)
- Only in fset: maps (42%), kset (44%), kset-native, intersection*, project*, select-key, select-keys*, index*
- Brewing in fset: powerset, cartesian, symmetric-diff
- Not in fset (just in clojure.core): set, ordered-set, hash-set, disj, set?

DEMO

- https://github.com/reborg/fset-talk/tree/master/20210706-lndclj/demo

APPROACH TO PERFORMANCE WORK

- Inner cycle
  - Ensure correctness (tests or other invariants)
  - Measure to establish the baseline
  - Introduce a single set of improvements
  - Repeat from top to establish new baseline
- Outer cycle
  - Depart from current design (if necessary)
  - For example, use alternative data structures
  - Or different algorithms

INNER CYCLE TOOLS (IN DESCENDING ORDER OF UGLINESS)

- Fix reflection warnings (if any) adding type hints
- Go transients
- Several degrees of "descending into Java”
  - Call methods directly (prefer interfaces when possible)
  - Replace reduce with loop iteration
- Specific arities, avoiding the catch-all varargs
- Possibly other uglier ad-hoc tricks:
  - Rely on specific input types, restricting on generality
  - Redefine hashcode() semantic in hash-maps (assuming hotspot)

TOOLS

- Criterium (or similar)
- VisualVM or equivalent (for explorative understanding)
- Javap (or equivalent, for low level understanding)
- clojure.core/time (sparingly, mainly macro-benchmarks)
- End to end in real-life project (if possible)

SPEEDING UP UNION: BASELINE

```
(defn union-0
  "This is the same as clojure.set/union with just one arity."
  [s1 s2]
  (if (< (count s1) (count s2))
    (reduce conj s2 s1)
    (reduce conj s1 s2)))

(let [s1 (set (range 200))
      s2 (set (range 100 300))]
  (quick-bench (union-0 s1 s2)))
;; Execution time mean : 26.754516 µs
;; this is our baseline
```

SPEEDING UP UNION-1

```
(defn union-1
  "One of the arguments is repeatedly mutated.
  Let's go transient."
  [s1 s2]
  (persistent!
    (if (< (count s1) (count s2))
      (reduce conj! (transient s2) s1)
      (reduce conj! (transient s1) s2))))

(let [s1 (set (range 200))
      s2 (set (range 100 300))]
  (quick-bench (union-1 s1 s2)))
;; Execution time mean : 20.209692 µs
;; Doing better.
```

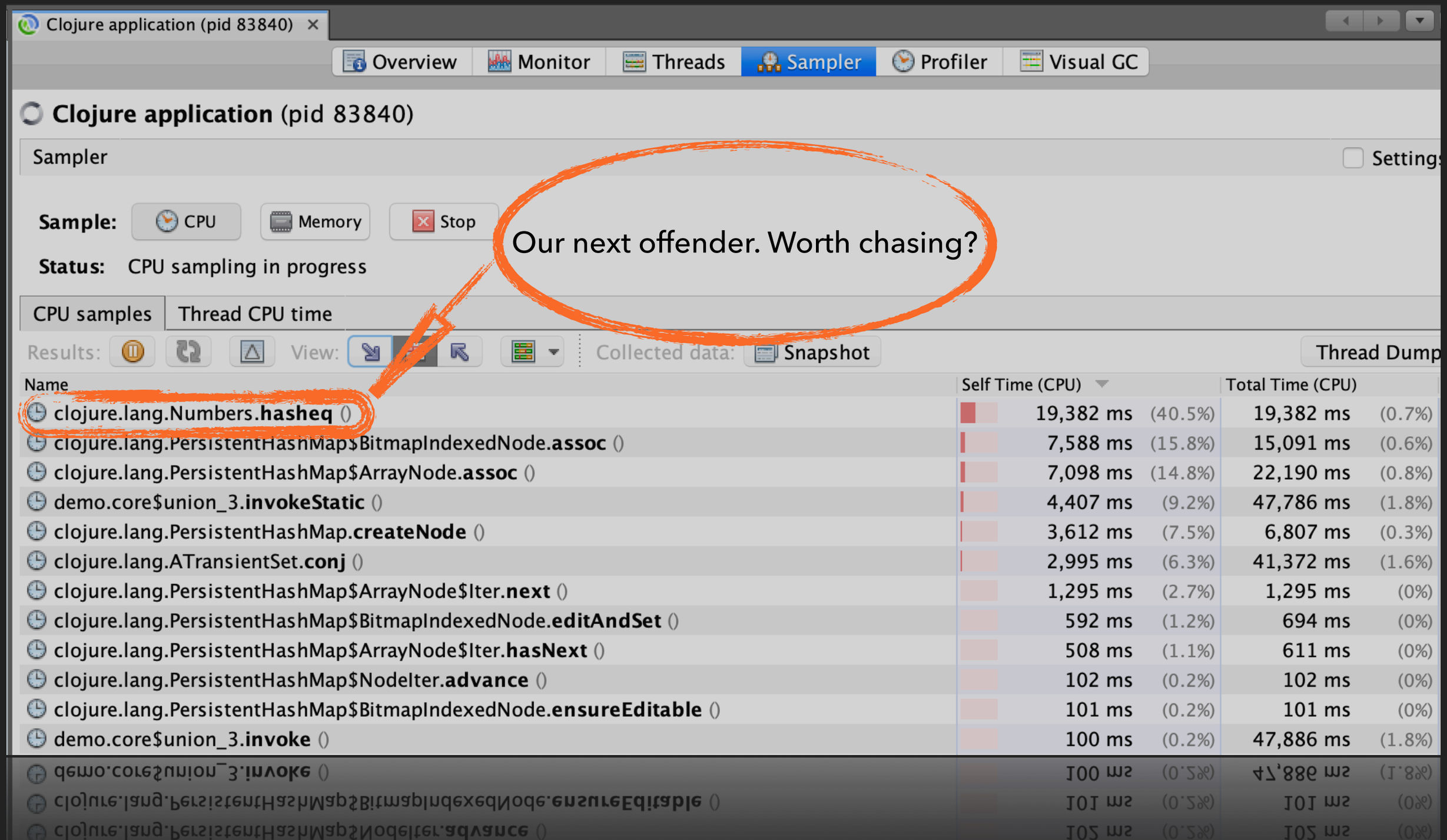
SPEEDING UP UNION-2

```clojure
;; Time to set this on.
(set! *warn-on-reflection* true)

(defn union-2
  "If it's only set we are talking about, can we take
  advantage of this information? We can then use Java interop."
  [^IEditableCollection s1 ^IPersistentSet s2]
  (if (< (count s1) (count s2))
    (recur s2 s1)
    (.persistent ^ITransientSet
      (reduce
        (fn [^ITransientCollection s item]
          (.conj s item))
        (.asTransient s1)
        s2))))

(let [s1 (set (range 200))
      s2 (set (range 100 300))]
  (quick-bench (union-2 s1 s2)))
;; Execution time mean : 19.038790 µs
;; Meh, small increment for all that noise.
```

SPEEDING UP UNION-3

```
(defn union-3
  "Let's get rid of reduce, we can replace it with a loop."
  [^IEditableCollection s1 ^IPersistentSet s2]
  (if (< (count s1) (count s2))
    (recur s2 s1)
    (let [^Iterator items (.iterator ^Iterable s2)]
      (loop [^ITransientSet s (.asTransient s1)]
        (if (.hasNext items)
          (recur (.conj s (.next items)))
          (.persistent s))))))

(let [s1 (set (range 200))
      s2 (set (range 100 300))]
  (quick-bench (union-3 s1 s2)))
;; Execution time mean : 15.361588 µs
;; Interesting. Looping is improving substantially.
;; What's next? Any guess?
```

SPEEDING UP UNION: FIRE UP THE PROFILER



SPEEDING UP UNION: NEXT STEPS

- Hotspot: calculating hashes for items to conj into the set.
- Structural problems likely require drastic design changes:
  - If you know the type of the items, create your own “Box” wrapper to override hashcode()
  - Take advantage of integer bit-wise operations (assuming integer sets only!)
  - Use different set implementation, go mutable, but that defeats the purpose of most of Clojure principles

SPEEDING UP UNION: LESSON LEARNED

- You might have to compromise generality/elegance with speed
- Always measure: some changes look promising when they are not
- The code gets uglier quickly the more you push it for speed
- When out of ideas, fire up the profiler or stare at the code!
- When possible, delegate to a library:
  - The code gets uglier, requiring specific knowledge
  - Might need to track changes to clojure.core or the JVM
  - Let someone else maintain that for you, if possible.

TRYING FSET IN THE WILD

DATASCRIPT

	add-1	add-5	add-all	init	retract-5	q1	q2	q3	q4	qpred1	qpred2	freeze	thaw
before	650.2	913.1	878.3	33.4	622.2	2.4	6.2	9.6	15.0	8.8	31.1	823.1	1995.8
after	651.2	901.8	861.3	28.3	625.3	2.1	6.1	9.1	14.3	8.8	30.7	833.5	1965.7

- Test env: version 1.1.0, SHA ae62fa6, openjdk version "11.0.9.1"
- Datascript uses `clojure.set` in a few critical sections
- It comes equipped with a handy set of benchmarks
- Verdict: promising speedups on querying benchmarks
- Most of them not cljs compatible (as fset is not).

TRYING FSET IN THE WILD

CRUX

```
       | run-tpch-queries
before | 339841.871333 ms
after  | 291307.660398 ms
```

- Test env: version 1.17.1, SHA 11fd8257, openjdk 11.0.9.1
- Crux query engine uses `clojure.set` mostly at compilation
- Still part of the API, although not sitting on a hot path
- Verdict: interesting, Crux team to investigate adoption

RIEMANN

```
       | indexing         | expiring
before | 28.775671 ms     | 13.143244 ns
after  | 26.060983 ms     | 14.106172 ns
```

- Test env: version 0.3.7, SHA 2d590cf, openjdk 11.0.9.1
- Riemann is a popular distributed monitoring system.
- It depends on `clojure.set` in some core parts.
- Verdict: inconclusive.

GOOD TO KNOW: CLOJURE.SET IS PRONE TO SILENT “GIGO”

```
(require '[clojure.set :as sets])

(sets/union #{1 2 3} [3 4])
;; #{1 4 3 2}
(sets/union #{1 2} [2 3 4])
;; [2 3 4 1 2]

(sets/difference #{1 2} #{2 3 4})
;; #{1}
(sets/difference #{1 2} [2 3 4])
;; #{}

(sets/superset? nil #{})
;; true
(sets/subset? #{0 3} [:a :b :c :d])
;; true
```

USEFUL LINKS

- https://github.com/droitfintech/fset enjoy!
- https://livebook.manning.com/book/clojure-the-essential-reference/chapter-13 the set chapter on my book (shameless plug)
- Slides: https://github.com/reborg/fset-talk/tree/master/20210706-lndclj/slides
- Get in touch: **@reborg -** reborg@reborg.net
- Support London Clojurians! https://opencollective.com/london-clojurians

THANKS!
QUESTIONS?