Efficient, scalable caching using ESQL shared variables in WMB and IIB

Emir Garza (emir.garza@prolifics.com)
Tim Dunn (tim_dunn@uk.ibm.com)
Gerardo Brenner (gerardo.brenner@prolifics.com)

Premier Business Partner IBM

Conventional Caching using ESQL

- The most common cache implementation with ESQL shared variables consists of a shared ROW that contains the result of a SELECT on the database table being cached:

```
DECLARE CACHE SHARED ROW;
```

- For example, a database table called “AIRPORTS” contains two columns, “CODE” and “CITY”. This code loads the cache:

```
SET CACHE.AIRPORT[] = SELECT A.CODE, A.CITY FROM
                      Database.AIRPORTS AS A;
```

- The CACHE variable will be populated like this:

```
CACHE.AIRPORT[1].CODE = AAA
CACHE.AIRPORT[1].CITY = Anaa
CACHE.AIRPORT[2].CODE = AAB
CACHE.AIRPORT[2].CITY = Arrabury
```

Conventional Caching using ESQL (Cont.)

- This function implements the cache:

```
CREATE PROCEDURE getCity_v01 (IN airportCode CHARACTER)
RETURNS CHARACTER
BEGIN
-- PERFORMANCE TEST ONLY! No ATOMIC blocks.
-- Do not use if Additional Instances > 0.
IF CACHE.AIRPORT.CODE IS NULL THEN
-- load the cache
   SET CACHE.AIRPORT[] = SELECT A.CODE, A.CITY FROM
                         Database.AIRPORTS AS A;
END IF;
RETURN THE(SELECT ITEM A.CITY FROM CACHE.AIRPORT[] AS A
           WHERE A.CODE = airportCode);
END;
```

Conventional Caching using ESQL – The Problem

- The problem with this cache structure is that it doesn't scale. A user trace will show that SELECT scans the table sequentially until it finds a row that satisfies the WHERE clause. As the table grows, the search gets slower. There comes a point when it's faster to drop the cache and go to the database each time.

- Measurements conducted by the authors (IBM Integration Server V9 on Windows 7 64 bit, with a local DB2 10.1 database) show the effect of a growing cache:

Conventional Caching using ESQL – The Problem

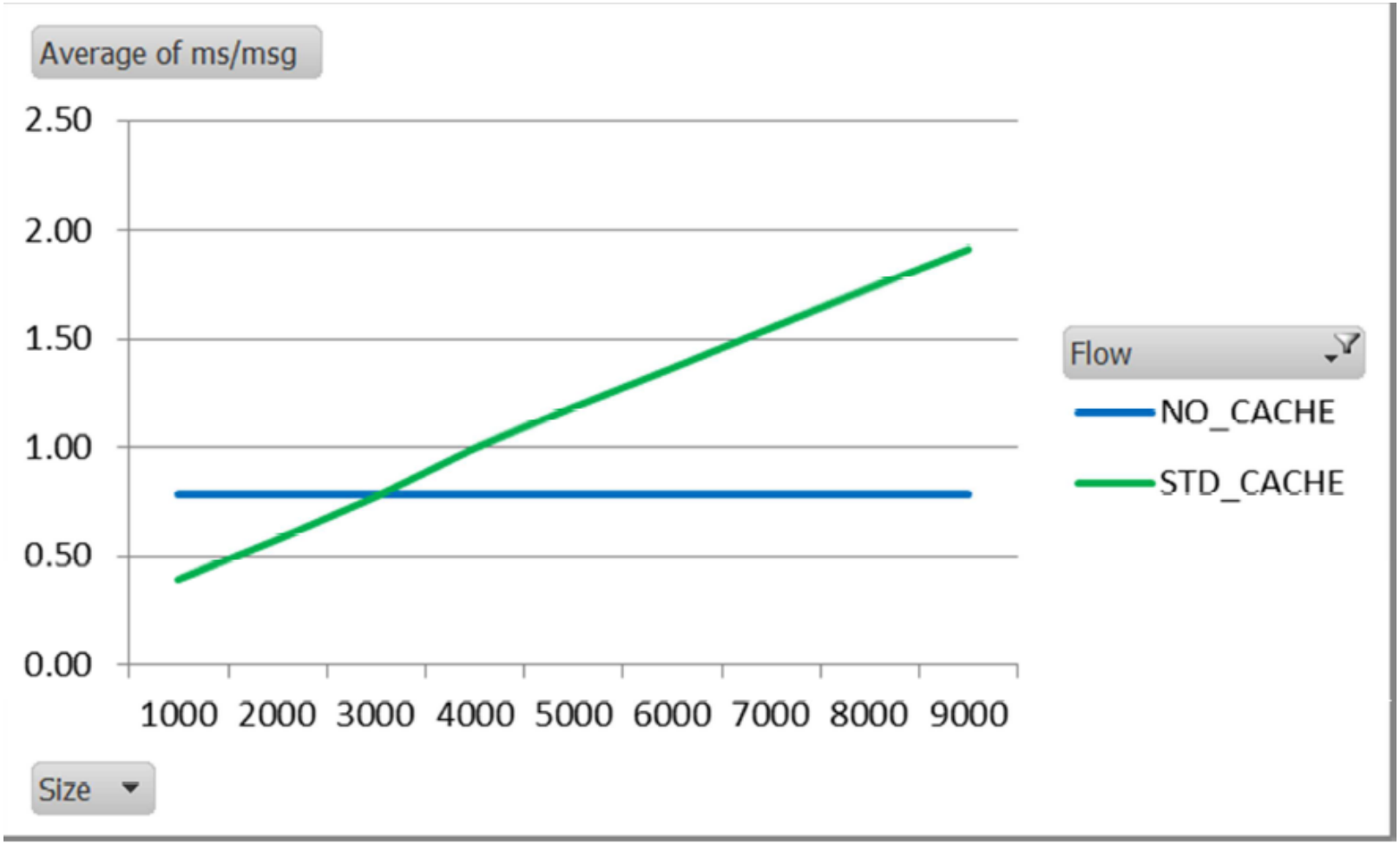

Figure 1: Elapsed Milliseconds per Message by cache size (no cache vs. standard cache)

ESQL cache - New Method

- The new cache stores each key and value (in our example, airport code and city name) as a NAMEVALUE pair:

```
CACHE.AAA = Anaa
CACHE.AAB = Arrabury
...
CACHE.ZZV = Zanesville
```

- Note there is no array. To return the city name for a given an airport code, the cache search function simply refers to the appropriate variable:

```
RETURN CACHE.{airportCode};
```

ESQL cache - New Method (Cont.)

```
CREATE PROCEDURE getCity_v02 (IN airportCode
CHARACTER) RETURNS CHARACTER BEGIN
IF NOT EXISTS(FIELDNAME(CACHE.*[]) THEN
   DECLARE TEMPCACHE ROW;
   SET TEMPCACHE.AIRPORT[] = SELECT A.CODE, A.CITY
         FROM Database.AIRPORTS AS A;
   FOR cacheline AS TEMPCACHE.AIRPORT[] DO
      CREATE LASTCHILD OF CACHE NAME cacheline.CODE
      VALUE cacheline.CITY;
   END FOR;
END IF;
RETURN CACHE.{airportCode};
END;
```

ESQL cache - New Method (Cont.)

- Because the search accesses the variable directly, it is much faster and scales better. The chart below compares the response time of the ESQL cache with the standard cache. The plot shows milliseconds per message, for cache sizes up to 9,000 rows:

ESQL cache - New Method (Cont.)

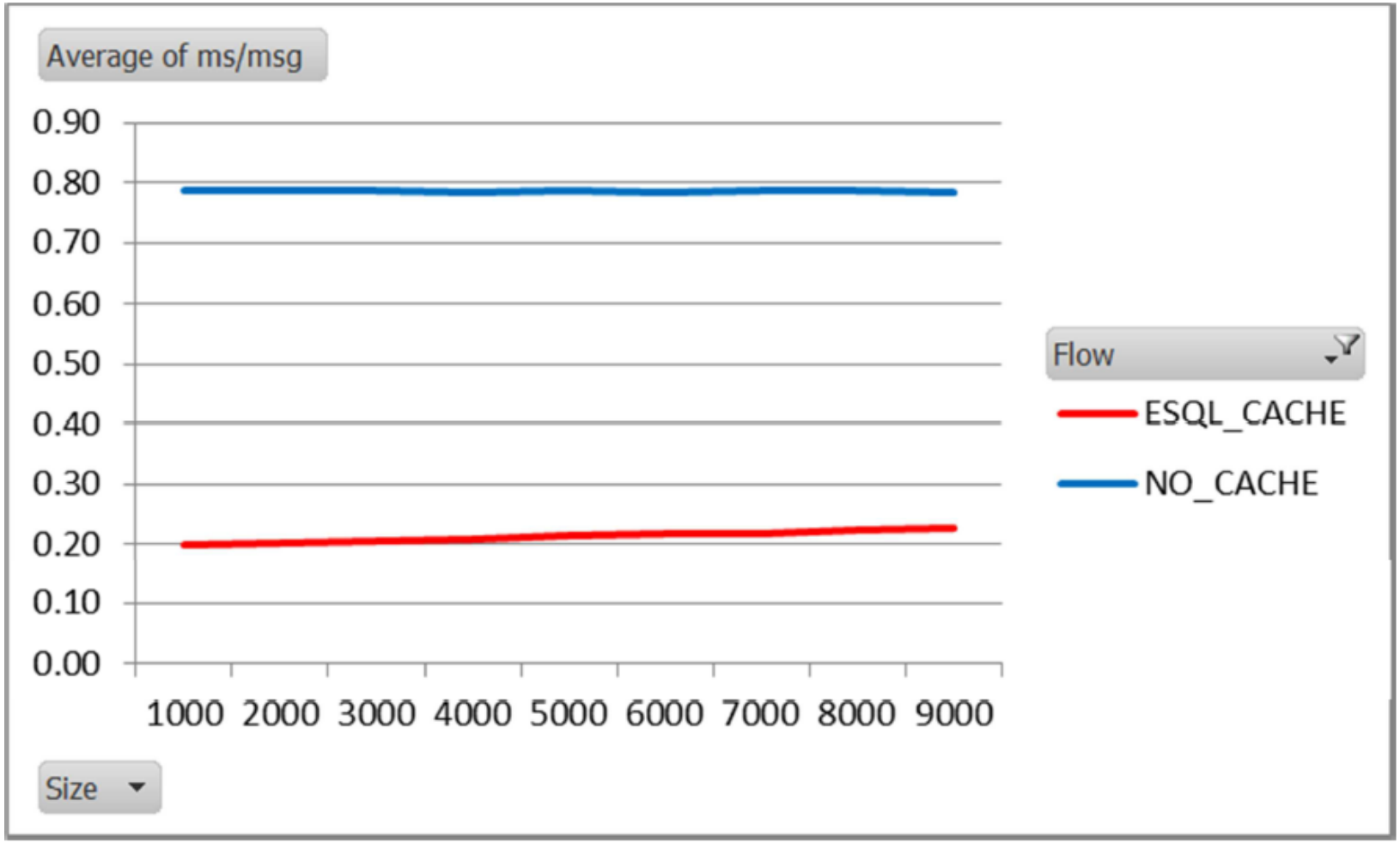

Figure 2: Elapsed Milliseconds per Message by cache size (ESQL cache vs. no cache)

Comparison with Global Cache

Message Broker Version 8 introduced Global Cache.

It uses the Execution Group's JVM (more specifically, the JVM's heap) to store data, and provides Java APIs to put to and get from the cache. It is easy to implement and provides consistent performance across a range of cache sizes. One advantage of the Global Cache over ESQL shared variables is that the cache can be shared between message flows, integration servers / execution groups, and integration buses / brokers (recall that the scope of ESQL shared variables is the message flow).

The next table shows how Global Cache compares with ESQL Cache:

Comparison with Global Cache (Cont.)

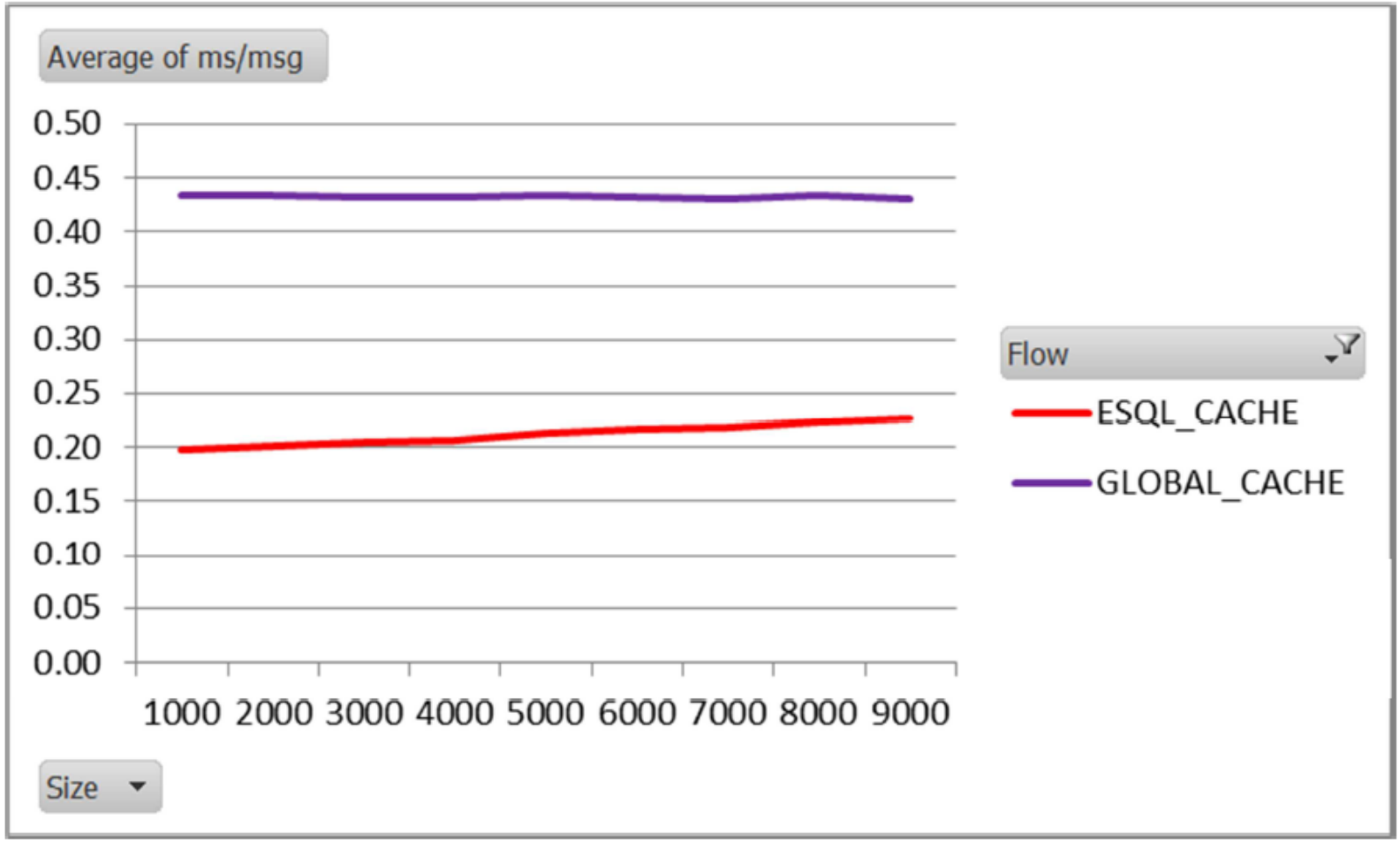

Figure 5: Elapsed Milliseconds per Message by cache size (ESQL cache vs. Global Cache)

Conclusion

- The proposed cache structure results in a significant performance improvement for caching with ESQL shared variables.
- The logic to implement the new cache is very simple, so converting existing standard cache structures should be straightforward.
- The authors have not measured cache sizes beyond 9,000 rows, but the trend indicates that a cache of 80,000 entries could still be faster than accessing the database (the precise cutoff point will be different for different hardware and software configurations). This is a significant improvement over the current cutoff point of a few thousand entries.

Conclusion (Cont.)

- For larger cache sizes, consider using the Global Cache, as it provides consistently good performance and is easy to implement. If it is necessary to share the cache between message flows, integration servers / execution groups or integration buses / message brokers, then the Global Cache is the only option, as the scope of ESQL shared variables is the message flow.

References

- The new caching mechanism has been implemented during development work at a big South Africa bank (IBM), as well as an UK motoring organization (Prolifics).
- The full article about Efficient, scalable caching using ESQL is about to be published in DevelopmentWorks