Spreadsheets for Stream Partitions and Windows

Position Paper

Martin Hirzel, Rodric Rabbah, Philippe Suter, Olivier Tardieu and Mandana Vaziri
IBM T.J. Watson Research Center
Yorktown Heights, NY 10598
{hirzel,rabbah,psuter,tardieu,vaziri}@us.ibm.com

***Abstract*—We discuss the suitability of spreadsheet processors as tools for programming streaming systems. We argue that, while spreadsheets can function as powerful models for stream operators, their fundamental boundedness limits their scope of application. We propose two extensions to the spreadsheet model and argue their utility in the context of programming streaming systems.**

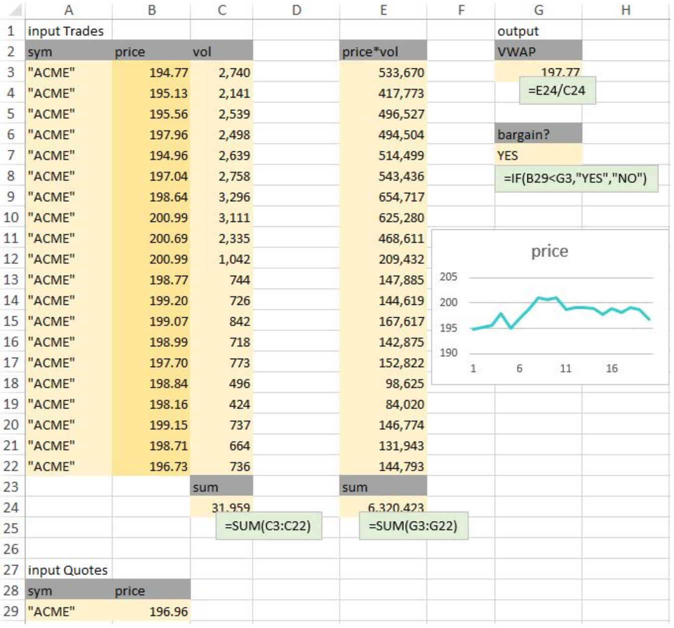
I. STREAM PROCESSING WITH SPREADSHEETS

Stream processing is an appealing approach to problems involving high volumes of data. Such problems arise in many different domains: finance, health care, consumer analytics, telecommunications, etc. Streaming applications are typically structured as a network of independent processing operators that consume and produce *tuples* of data. This architecture enables large-scale deployments, as operators can be distributed and replicated across machines for scalability, maximizing data-parallelism and thus resulting in systems that can handle millions of tuples per second. It also naturally enforces a clear separation of concerns: because operators only need to know the format of incoming and outgoing tuples, their implementations are largely independent of each other and they can even be developed by separate teams.

We argue that spreadsheets are particularly well suited for programming operators in streaming applications. Indeed, spreadsheet and stream programming systems share many important commonalities:

- **Two-dimensional view of data**: spreadsheets organize data in rows and columns, and streams are most naturally modelled as a sequence (rows) of tuples of attributes (columns).
- **Reactive model of computation**: a change of value in a spreadsheet cell results in the immediate propagation of computations to all dependent cells. Similarly, stream operators produce new tuples as a reaction to inputs.
- **Stateless computations**: streaming systems embrace statelessness as it facilitates encapsulation and thus parallelization. Spreadsheets favor statelessness to maintain a model of idempotent (re-)computations.

A final point is simply pragmatism: domain experts are not always programmers, but they will typically be familiar with spreadsheet processors, as more than 500 million people worldwide are reported to use spreadsheets [1]. Supporting stream programming within a familiar tool thus removes the need for an additional expert programmer and the associated translation issues.



	A	B	C	D	E	F	G	H
1	input Trades						output	
2	sym	price	vol		price*vol		VWAP	
3	"ACME"	194.77	2,740		533,670		197.77	
4	"ACME"	195.13	2,141		417,773		=E24/C24	
5	"ACME"	195.56	2,539		496,527			
6	"ACME"	197.96	2,498		494,504		bargain?	
7	"ACME"	194.96	2,639		514,499		YES	
8	"ACME"	197.04	2,758		543,436		=IF(B29<G3,"YES","NO")	
9	"ACME"	198.64	3,296		654,717			
10	"ACME"	200.99	3,111		625,280			
11	"ACME"	200.69	2,335		468,611			
12	"ACME"	200.99	1,042		209,432			
13	"ACME"	198.77	744		147,885			
14	"ACME"	199.20	726		144,619			
15	"ACME"	199.07	842		167,617			
16	"ACME"	198.99	718		142,875			
17	"ACME"	197.70	773		152,822			
18	"ACME"	198.84	496		98,625			
19	"ACME"	198.16	424		84,020			
20	"ACME"	199.15	737		146,774			
21	"ACME"	198.71	664		131,943			
22	"ACME"	196.73	736		144,793			
23			sum		sum			
24			31,959		6,320,423			
25			=SUM(C3:C22)		=SUM(G3:G22)			
26								
27	input Quotes							
28	sym	price						
29	"ACME"	196.96						

Fig. 1. A simple trading bargain calculator.

With these considerations in mind, we developed ActiveSheets [2], a platform for developing streaming operators in Microsoft Excel. ActiveSheets integrates with IBM Infosphere Streams and the SPL programming language [3], allowing end-users to visualize live streaming data in spreadsheets, and to export their computation models as SPL operators.

As an illustration of programming with ActiveSheets, we consider a simple financial application to detect bargains in quoted stock prices, based on trading data. We observe a given stock symbol, with a history of trades at prices P_i and volumes V_i. The volume-weighted average price, a standard metric in finance, is defined as:

$$\text{VWAP} = \frac{\sum_i P_i \cdot V_i}{\sum_i V_i}$$

A quoted price is deemed a bargain if it is lower than the VWAP.

Figure 1 shows how we can detect bargains in ActiveSheets. Cells `A3:C22` are mapped to the input stream of executed trades and, as tuples come in, are updated to always reflect the last 20 tuples (a scrolling effect). This conforms with the natural view of streams as rows of tuples. Cells `A29:B29` contain the latest available quote, and are similarly updated, potentially asynchronously from trades. All other cells and graphics are standard Excel content, and update reactively to the incoming streams. The ActiveSheets user marks the result of the computations for export, in this case the cells G3 and G7, thus determining the outputs of the operator.

This programming experience is natural to users familiar with spreadsheets, and leads to rapid development of malleable models. A fundamental limitation of spreadsheets, however, is their bounded nature. In the following sections, we highlight how this affects the programming of streaming operators and propose potential extensions to alleviate these issues.

II. VARIABLE-SIZE WINDOWS

We first consider a limitation arising from boundedness in the vertical dimension. In the example described in the previous section, the computation of the VWAP happens over a window of fixed size (the *last* 20 *values*). A more realistic model would instead mandate aggregation over, e.g., the *last minute of trading data*. Such a window is problematic to represent in a spreadsheet, as its size in terms of tuples cannot in general be computed statically.

One approach (the current state of ActiveSheets) is to require application developers to compute aggregations over variable-size windows outside of the spreadsheet, in a separate set of operators, and to only stream the aggregated data into the spreadsheet. While it can be straightforward for problems as simple as VWAP, in general it takes away flexibility from the spreadsheet developer and limits the expressiveness of the models that can be developed. Another solution would be to impose a *maximum* size on variable-size windows. This would make it possible to allocate a sufficiently large number of cells in the spreadsheet in advance, but would not properly support scenarios where an unpredictably high number of tuples must be consumed in a short timespan.

We argue that a better approach is to introduce a new function to spreadsheets, WINDOW, to allow users to store a *list* of values in a single cell. The spreadsheet processor can enforce that such cells only flow into aggregating functions such as COUNT, SUM, AVERAGE, etc., producing #VALUE! errors otherwise. The aggregating functions are by design already capable of handling variable argument counts, and also have sensible default values when applied to empty ranges. Their semantics when applied to a window are thus predictable and natural.

Reactivity can be preserved by triggering recomputation only when a new value enters the window. This may go counter to the notion that the window should contain *exactly* the values for, say, the last minute, but we believe it matches the natural intuition that the value of a cell changes whenever its inputs do. Finally, many such aggregation functions can be implemented efficiently using incremental or partial recomputations, preserving the performance and responsiveness associated with spreadsheets.

III. KEY-BASED PARTITIONING

The second limitation we want to address is in handling unboundedness in data variation. In our example in Figure 1, we detect bargains for the ACME ticker symbol only, but what if we wanted to detect bargains for other symbols, or for all incoming symbols?

Quotes and trades for a specific symbol can be easily extracted from the streams of all quotes and trades using IF or MATCH expressions as filters. One approach is therefore to replicate the computation n-ways, selecting a particular symbol each time. While this approach is acceptable for a small and known number of symbols, it does not scale and requires defining a list of symbols to monitor a priori.

We argue that a better approach is to let users define and visualize the full computation for *one* defined symbol. To do so, they can filter the incoming stream with a special SELECT function, whose behavior is to update the value of the cell only if its argument matches an expected value. The rest of the computation is left as usual. Once the computation is fully defined, the programmer replaces SELECT by another new function, PARTITION. Visually, PARTITION behaves identically. The semantics, however, are that for each distinct value that flows into PARTITION, a conceptually independant instance of the spreadsheet performs the computation. We can think of programming with PARTITION as another instance of the paradigm, familiar to spreadsheets users, of programming by example.

In the VWAP example, the PARTITION function would be used on two streams: quotes and trades. To preserve ease of use and enable efficient implementation, we can require that all occurrences of the PARTITION function in a spreadsheet agree on the key, i.e., we can forbid partitioning the trades by ticker symbol and the quotes by, say, geography.

IV. CONCLUSION

Spreadsheets are a suitable medium for the development of operators in streaming applications. Their familiar interface and computation model make it possible for domain experts to participate in the development of complex applications. However, natural operations on streaming data such as windowing and partitioning are not directly expressible in spreadsheets due to their bounded nature. These limitations restrict the autonomy of application developers, thus partially defeating the purpose of the separation of concerns. We propose extensions to the spreadsheet model that expand the scope of expressible computations and that, we believe, adopt natural semantics. We plan to formalize our expanded computation model and to support the proposed extensions in a future version of ActiveSheets.

REFERENCES

[1] S. Gulwani, “Automating string processing in spreadsheets using input-output examples,” in *POPL*, 2011, pp. 317–330.

[2] M. Vaziri, O. Tardieu, R. Rabbah, P. Suter, and M. Hirzel, “Stream processing with a spreadsheet,” in *ECOOP*, 2014, pp. 360–384.

[3] M. Hirzel, H. Andrade, B. Gedik, G. Jacques-Silva, R. Khandekar, V. Kumar, M. Mendell, H. Nasgaard, S. Schneider, R. Soulé, and K.-L. Wu, “IBM Streams Processing Language: Analyzing big data in motion,” *IBM Jour. of Research & Dev.*, vol. 57, no. 3/4, pp. 7:1–7:11, 2013.