Обзор состояния области потоковой обработки данных

Р.С. Самарев <samarev@acm.org>
Московский государственный технический университет имени Н.Э. Баумана
105005, Москва, 2-я Бауманская ул., д. 5, стр. 1

Аннотация. В статье рассматриваются аспекты построения и использования современных фреймворков для организации потоковой обработки данных. Уделяется внимание архитектурным аспектам фреймворков, а также связанными с ними достоинствами и недостатками. Рассматривается проблема объективного оценивания характеристик потоковых фреймворков.

Ключевые слова: большие данные; тестирование; storm; spark; flink; samza

DOI: 10.15514/ISPRAS-2016-29(1)-13

Для цитирования: Самарев Р.С. Обзор состояния области потоковой обработки данных. Труды ИСП РАН, том 29, вып. 1, 2017 г., стр. 231-260. DOI: 10.15514/ISPRAS-2017-29(1)-13

1. Введение

Термин "потоковая обработка данных" (Stream processing) подразумевает обработку любых данных как потока, однако сейчас под ним принято понимать скорее обработку некоторых крупных пакетов данных, таких как некоторое сообщения, но не обработку аудио и видео данных. В этой статье мы будем говорить именно о состоянии области обработки больших данных как потоков, то есть об их обработке с минимальной задержкой.

Идеи потоковой обработки данных совершенно не новы. В 80-е годы они появились под терминами Dataflow processing, Dataflow database machine и пр. Однако в близком к современному пониманию первые работы в этой области стали появляться в конце 90-х – начале 2000-х годов на волне развития Интернет, появления идей Интернета вещей как итог понимания того, что традиционный способ обработки данных через сохранение в статичную базу данных не подходит для обработки постоянно обновляемых данных. Интересен один из обзоров в этой области, который был представлен в 2003-м году в работе [26]. Из числа первых потоковых процессоров и фреймворков (будем называть так программные продукты, которые предназначены для

потоковой обработки данных) Aurora, Borealis, COUGAR, Gigascop, NiagaraCQ, OpenCQ, StatStream, STREAM, TelegraphCQ, Tribeca ни один не получил коммерческого развития. (Аббревиатура CQ в названии проектов - это Continuous Query.) Более подробно см. в [19].

Чуть более подробно рассмотрим академический проект Aurora [16], который включал в себя довольно много новшеств для своего времени. Этот продукт позволял описать процесс непрерывной обработки данных, включая несколько параллельных входов и выходов, поддерживалась возможность оперативного накопления данных в локальном хранилище для их обработки. Модель программирования - визуальная, в терминах алгебры SQuAl ([S]tream [Qu]ery [Al]gebra). Поддерживались основные операции filter, map, union, sort, aggregate, join, resample. Описание процесса обработки данных представляет собой последовательноть блоков-операций и стрелок, означающих передачу данных, что образует направленный ациклический граф (DAG). Итогом этого проекта стала работа [40], в которой М. Стойнбрейкер, У. Гетинтемел и С. Здоник сформулировали требования к системам потоковой обработки реального времени.

Приведём эти требования, сформулированные в виде правил:

1. Сохраняйте данные движущимися.
2. Формулируйте запросы с использованием SQL на потоках (StreamSQL).
3. Справляйтесь с дефектностью потоков (задержка, отсутствие и нарушение порядка данных).
4. Генерируйте предсказуемые результаты.
5. Интегрируйте хранимые и потоковые данные.
6. Гарантируйте безопасность и доступность данных.
7. Автоматически разделяйте и масштабируйте приложения.
8. Мгновенно обрабатывайте и выдавайте результаты.

Как видим, несмотря на более чем десятилетний возраст публикации, эти требования ещё актуальны, хотя и с некоторыми оговорками, и не в полной мере реализуются в современных средствах потоковой обработки. Не все выполняют требование 2. В ряде случае проблемы с требованием 5, часто имеются проблемы с требованиями 6-8.

Близкой к области потоковой обработки данных является так называемая сложная обработка событий – CEP (Complex Event Processing). Она заключается в том, что на основании заданных правил обработки сообщений и их последовательности выявляются и обрабатываются некие события. Это направление традиционно отличается собственными языками написания правил, например, Drools [7]. Условно CEP отличают от потоковой обработки тем, что CEP обычно не являются массово параллельными системами обработки, они обеспечивают минимальную задержку времени обработки,

отсутствует необходимость программировать на традиционных языках программирования (часто достаточно SQL или событийного языка типа Drools) [19]. Также в CEP очень часто не требуется гарантированная обработка всех сообщений и в случае сбоя часть их можно отбросить. Однако принципиально CEP – это та же обработка потока сообщений. И, более того, современные потоковые фреймворки типа Apache Flink [2], Apache Kafka Streams [29] претендуют на возможность их использования в качестве средств CEP.

2. Типовая схема приёма и обработки потоковых данных

В данной статье рассматриваются, прежде всего, доступные программные продукты с открытым исходным кодом. Поэтому приводимые здесь схемы будут относиться именно к ним.

Одно из основных требований при обработке потоков – это возможность принять тот поток, который поступает на вход. Отметим, что в зависимости от решаемых задач, этот поток может быть постоянным или сильно изменяющимся во времени, предсказуемым или содержащим резкие всплески, на которые необходимо корректно реагировать. Следует отличать потоки от погодных сенсоров и, например, пиков активности пользователей сетевых игр, информацию о которых нам надо собрать.

Сложившаяся схема обработки выглядит примерно так, как это представлено на рис. 1.

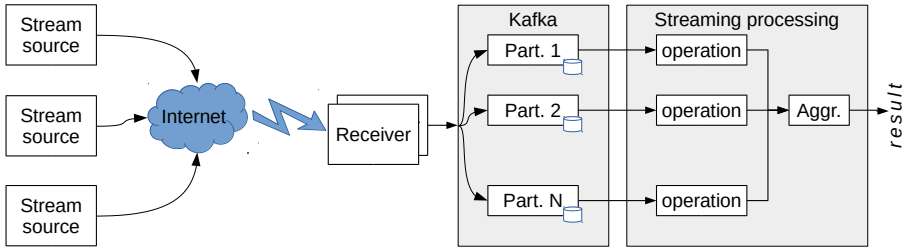

Рис. 1: Типовой пример сбора и обработки сообщений

Fig. 1: Typical process of data gathering and processing

Входные потоки, образованные внешними источниками, необходимо принять и упорядочить. Чаще всего для этого используют программный продукт Apache Kafka [3]. Однако заметим, что сейчас это правило уже может быть поставлено под сомнение, поскольку другие потоковые фреймворки решают сходные задачи самостоятельно. Основное назначение Apache Kafka – принять входные сообщения и гарантировать их доставку потребителям. Сообщения помещаются в очередь (topic), которая может быть сохранена в промежуточную БД, что гарантирует последующее получение данных потребителями, которые сейчас не активны. По умолчанию в качестве СУБД для промежуточной БД используется RocksDB. Дополнительно Kafka может

разложить сообщения по узлам кластера в соответствии с использованной функцией размещения. Это позволяет равномерно распределить дальнейшую вычислительную нагрузку. Поэтому следующий этап – использование соответствующего потокового фреймворка, который обеспечит размещение задач обработки данных, их координацию и чтение из очереди сообщений Kafka. Подробнее см. [28].

3. Основные различия потоковых фреймворков

Массовой проблема использования потоковых фреймворков стала лишь в последние годы. В отношении того, как должен выглядеть современный потоковый фреймворк, существует множество мнений. Приходится наблюдать, что то, о чём писали буквально год-два назад, уже стало устаревшим, а выглядевшие год назад истинными утверждения стали ложными.

Современное понимание потоковых процессоров изложено в статьях [37, 38]. На первом месте выделяют различия в обработки данных и модели программирования.

3.1 Модель обработки данных

С точки зрения модели обработки данных выделяют системы естественной потоковой обработки, в которых исходные сообщения поступают на обработку строго последовательно. К этой группе относятся продукты Apache Storm [6], Apache Samza [4], Apache Flink [2]. Другой вариант обработки – пакетная (micro-batches). Она отличается тем, что перед тем как попасть на обработку, сообщения группируются в пакеты. К этой группе относятся Apache Storm/Trident и Apache Spark [5].

Ещё год назад публиковались статьи [34], где пакетная модель обработки демонстрировалась как достоинство. Проблема же заключается в особенности реализации конкретного фреймворка, и ответ на то, какая модель является предпочтительной, не является очевидным. Например, по состоянию на 2016 год, программы, использующие только Apache Storm, работают существенно быстрее, чем использующие его пакетную надстройку Apache Trident. Однако Apache Spark с пакетной моделью демонстрирует существенно большую производительность, чем Apache Storm, но также существенно проигрывает Apache Flink с естественной потоковой моделью обработки.

3.2 Модель программирования

Следующим аспектом использования фреймворков является модель программирования. Различают композиционную и декларативную модели. Композиционная модель подразумевает объединение элементарных операторов и источников данных в некоторую сетевую "топологию", то есть фактическое соединение создание элементов обработки и соединение их входов и выходов. Этой модели придерживаются Apache Storm и Apache Flume [12]. Декларативная модель в современном понимании предполагает

использование высокоуровневых операций, предписывающих определённый тип обработки данных. По сути, это напоминает функциональный стиль программирования. В следующем листинге приведён пример программы вычисления количества слов, реализованный с помощью Apache Flink.

```
StreamExecutionEnvironment env =
    StreamExecutionEnvironment
        .getExecutionEnvironment( );
DataStream<Tuple2<String,Integer>> dataStream=
     env.socketTextStream ("localhost", 9999)
        .flatMap (new Splitter ( ) )
        .keyBy( 0 )
        .timeWindow( Time.seconds ( 5 ) )
        .sum( 1 );
dataStream.print( );
env.execute ( "example")
```

Модель называется декларативной, потому что на момент выполнения этого кода, по сути, лишь выстраивается схема обработки потока, но ещё не происходит какой-либо реальной обработки данных. Это позволяет делать описание модели в общем-то на любом языке программирования при условии, что будут вызваны соответствующие методы класса, конструирующего этот логический план обработки. Это может быть специализированный язык предметной области или, например, может использоваться аналогичный код, написанный на языке Ruby и выполняемый в окружении JRuby с подключением соответствующих библиотек фреймворка. Или же может быть реализован транслятор xml-спецификаций по аналогии с тем, как организована модель спецификаций соединения модулей в Spring Framework [10]. Ограничением является лишь наличие соответствующих операторов обработки в терминах классов Java, если говорить о фреймворках Apache Flink или Apache Spark. В примере таким классом является класс Splitter, реализация которого не приводится. С использованием этого интерфейса могут быть реализовано и выполнение SQL-запросов. Также отметим тенденцию последних лет – создание специальных средств типа проекта Emma [18] или Apache Beam, назначением которых является автоматическая трансляция в вызовы именно того фреймворка, который выбрал пользователь-программист в данный момент.

Следует добавить несколько слов относительно изменений в понимании того, какие языки должны быть использованы для описания процесса обработки данных. Как уже упоминалось ранее [19], развитие потоковых фреймворков началось как ветвь области СУБД, поэтому изначально описание процесса обработки данных рассматривалось только с использованием SQL. Дальнейшее развитие, совпавшее с переходом от консольного интерфейса пользователя к графическому и соответствующей эволюцией идей программирования на уровне схем, привело к появлению проектов типа Aurora [16], где схема обработки была именно графической.

В 2000-е годы можно видеть массовый переход к графическим методам программирования. Это и появление языка BPMN, представляющего собой графические схемы бизнес-процесса, и массовое использование его модификаций в средствах ESB и ETL, и появление средств типа KNIME, которые позволяют описать процесс обработки данных как графическую схему. Очевидно, что использование графического языка программирования является привлекательной альтернативой SQL. Безусловно, нельзя забывать о языке описания правил типа Drools, однако его использование не было массовым.

Конец этого периода характеризуется всплеском интереса к функциональным языка программирования и к функциональному стилю программирования как таковому. Существенное влияние оказала популярность платформы JVM, на которой созданы такие языки, как Scala и Clojure. Сейчас функциональный стиль реализован и в самом языке Java. Вероятно, поэтому основной способ программирования для потоковых фреймворков – декларативный, в функциональном стиле. Отметим, что попытки вернуться к SQL были актуальны всё время, поэтому разработчики современных потоковых фреймворков заявляют о таких возможностях. Это относится к Apache Flink, Apache Spark, Apache Kafka Streams. Реально же всё это требует отдельной оценки.

В настоящий момент можно заключить, что именно декларативное программирование в функциональном стиле с использованием языка Java 8 является наиболее распространённой практикой при создании приложений с использованием потоковых фреймворков. Остальные языки программирования являются скорее нишевыми. Всплеск популярности языков типа Scala и Clojure следует считать временным явлением. Если пару лет назад Apache Spark позиционировался как фреймворк на Scala для Scala-программистов, то сейчас существенная часть его интерфейсного кода продублирована на Java для того, чтобы не терять клиентов и иметь возможность привлекать разработчиков, желающих использовать Java 8. Apache Storm пытались реализовывать на языке Clojure, но сейчас принято решение продолжать развитие фреймворка на языке Java.

3.3 Гарантированность обработки сообщений и устойчивость к сбоям

Следующей особенностью потоковых фреймворков является метод гарантирования обработки сообщений. Это обусловлено их распределённой архитектурой и необходимостью согласования данных на разных узлах кластера, что напрямую влияет на производительность. Различают обработку "максимум один раз" (at most once), "по крайней мере один раз" (at least once) и "строго один раз" (exactly once).

"Максимум один раз" означает, что сообщение может быть доставлено ноль или 1 раз, то есть может быть потеряно. "По крайней мере один раз"

предполагает, что сообщение будет доставлено на обработку в любом случае, но может быть более 1 раза. Очевидный недостаток этого метода – необходимость учитывать возможность появления дубликатов и, соответственно, нагрузку на программиста по их учёту в своих алгоритмах. "Точно один раз" означает, что сообщение будет гарантированно доставлено, но при этом исключены дубликаты. Этот метод самый удобный для программиста, но и, очевидно, самый сложный и медленный в реализации потокового фреймворка.

Очевидным требованием к распределённой системе обработки данных является требование корректной обработки в случае выхода из строя одного или нескольких узлов кластера. Обзор нескольких методов восстановления приводится в [30].

Одним из простейших механизмов восстановления обладают Apache Storm и Twitter Heron [13]. Этот механизм основан на отправке подтверждения каждым следующим оператором предыдущему. В терминологии Storm имеются источники данных (spout) и операторы обработки (bolt). Для каждой отправляемой записи (tuple) генерируется случайное 64-х битное число. После получения записи оператор обработки отправляет сообщение о получении. Если это сообщение не получено, исходная запись будет отправлена ещё раз. Следует заметить, что при этом существует проблема поддержки целостности данных в распределённой системе. Кроме того, подобный механизм является медленным и может порождать дубликаты [8].

Очевидной идеей ускорения является метод микропакетов, который использован в Apache Storm Trident и Apache Spark Streaming. Идея заключается в том, чтобы подтверждать не каждую запись, а небольшой пакет записей, если оператор выполнил их обработку (см. рис. 2). Это действительно позволяет существенно ускорить обработку и гарантировать доставку строго одного сообщения, однако создаёт следующие проблемы.

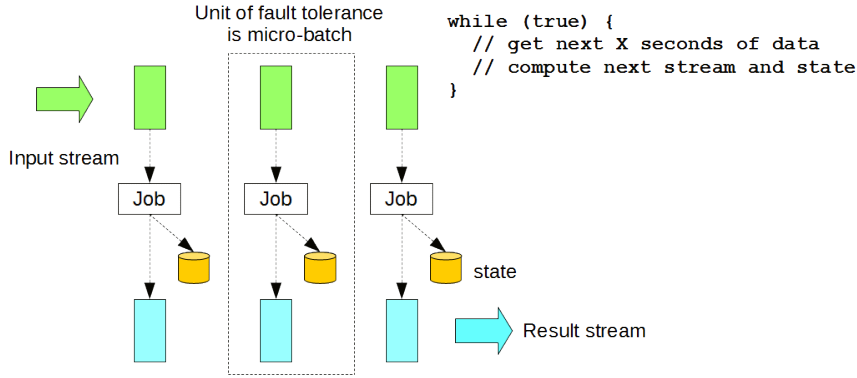

Рис. 2: Схема формирования контрольных точек с использованием микропакетов
Fig 2. Checkpointing of microbatches

В потоковых задачах часто требуется создание скользящего окна. Это окно может быть ограничено временем обработки, диапазоном времени сообщений или количеством сообщений. В случае же использования микропакетов нет возможности обрабатывать строго заданное количество сообщений. Другой серьёзный недостаток – сложность контроля темпа обработки в конвейере, образованном цепочкой операторов. То есть нет возможности контроля обратного давления данных. И последней проблемой является рост задержки обработки данных, поскольку подтверждение обработки будет получено лишь после обработки всего пакета. Подробнее см. [30].

Альтернативой микропакетам является использование транзакционного журнала. Этот подход применяется в Google Cloud Dataflow и Apache Samza. Основная идея – факт обработки сообщения фиксируется в журнале. В случае сбоя восстановление состояния возможно на основе записей в этом журнале. Такой подход позволяет решить большинство проблем предыдущих моделей, позволяя рассматривать поток сообщений и их обработку как непрерывный процесс, обеспечивая при этом высокую скорость обработки. Однако, например, в Apache Samza не решена проблема дубликатов сообщений.

В Apache Flink используется ещё одна модель восстановления, называемая авторами Asynchronous Barrier Snapshotting (ABS). Суть этой модели описана в [21]. Идея заключается в том, чтобы сохранять контрольные точки обработки сразу для последовательности сообщений. Имеются определённые точки снимков состояний – барьеры. В отличии от микропакетов, здесь процесс обработки не прерывается.

В момент объединения данных, получаемых от разных операторов, происходит так называемое выравнивание барьеров и запуск данных на обработку с формированием очередной метки-барьера в выходном потоке (см. рис. 3. Несмотря на то, что оператор может блокировать определённые входы до момента выравнивания барьеров, то есть появления методов-барьеров на всех входах, невозможна ситуация, когда оператор ничего не обрабатывает (хотя бы с одного входа данные поступают, иначе там уже присутствует метка-барьер), в отличии от микропакетов, где состояние простоя оператора вполне допустимо.

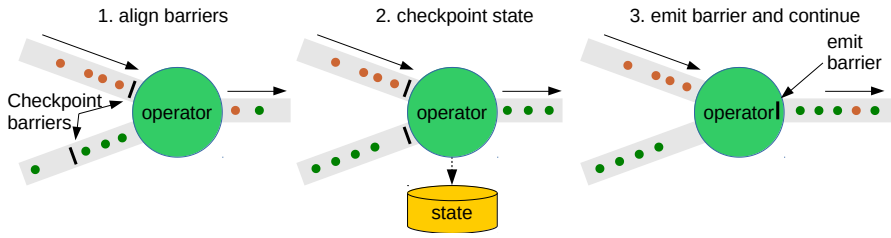

Рис. 3: Схема формирования контрольных точек Asynchronous Barrier Snapshotting

Fig. 3. Data processing with Asynchronous Barrier Snapshotting

3.4 Сохранение состояния и операции с окном данных

Для выполнения агрегационных запросов важно наличие возможности сохранение состояния со стороны приложения, выполняемого фреймворком. Под сохранением состояния здесь понимается возможность сохранять данные между обработкой сообщений, например, накопление данных в аккумуляторе для вычисления среднего значения. Другими словами, фреймворк без хранения состояний не может выполнить агрегацию данных.

Большая часть современных фреймворков поддерживает сохранение состояния, включая Apache Storm версии 1.0. Однако существуют различия в программировании накопления состояния, поскольку в зависимости от фреймворка это может быть локальное состояние данного оператора на данном узле, автоматически синхронизируемое глобальное состояние или явно разделяемая переменная, значение которой будет автоматически рассылаться по всем узлам кластера, где производится обработка.

В отношении оконных операций с данными следует отметить то, что в зависимости от фреймворка, поддерживается несколько типов окон: окна на фиксированное количество сообщений или на время. Время может быть системным или событийным, то есть отсчитываться на основе времени поступления сообщений в обработку или на основе времени, записанного внутри сообщений.

Также различают окна, скользящие с перекрытием данных, и последовательные окна обработки, где следующее окно начинается сразу после окончания предыдущего. Вариант окна с перекрытием позволяет, например, установить обработку данных за 1 минуту с выдачей результата раз в 5 секунд.

Как уже отмечалось ранее, реализуемость тех или иных типов окон зависит от реализованной во фреймворке модели гарантирования обработки сообщений.

Отметим, что в большинстве случаев хранение состояния технически реализуется с помощью встраиваемой СУБД RocksDB [9].

3.5 Распределение данных по узлам кластера и модель распараллеливания

Аспект разделения данных по узлам кластера и модель управления распараллеливанием операций является одним из существенных аспектов выбора фреймворка.

Следует сразу разделить фреймворки, способные выполнять операции распараллеливания автоматически, и фреймворки, не способные это делать.

К первой группе относится большинство потоковых фреймворков. Для примера на рис. 4 продемонстрирован процесс размещения и запуска приложения для Apache Flink. Необходимо выполнить команду размещения приложения на кластере под управлением уже запущенного Apache Flink, после чего построение физического плана выполнения, загрузка кода

операторов на соответствующих узлах, балансирование загрузки будут выполняться автоматически.

Ко второй группе можно отнести Apache Kafka Streams и Java Reactor. Особенность Apache Kafka Streams заключается в том, что приложение, написанное с помощью этого фреймворка, является автономным Java-приложением, выполняемом на одном узле (см. рис. 5). Фреймворк не предоставляет никаких средств передачи данных на уровне операций. Если требуется обмен данными с другими операторами, это означает, что данные следует выгрузить в очередь Kafka с необходимой функцией распределения данных, и должно быть написано ещё одно приложение, принимающие эти данные и продолжающее их обработку. То есть Kafka становится средством, обеспечивающим гарантированность обработки данных и их доставки по графу операторов, однако проблемой программиста является создание нужного количества приложений, их распределение по узлам кластера, а также запуск и мониторинг их состояния.

Отметим, что прямым конкурентом Apache Kafka Streams следует считать Java Reactor [15] и её подсистему Flux. Несмотря на то, что это средство не позиционируется как потоковый процессор, а создано как системный компонент Spring Framework и Java 9, Java Reactor обеспечивает выполнение типового набора операций над потоком сообщений, включая операции агрегации данных в окне.

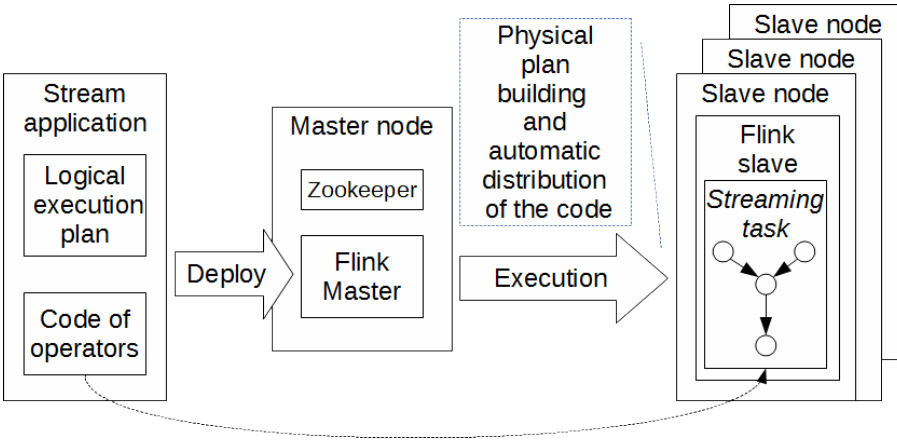

Рис. 4: Схема запуска приложения для Apache Flink
Fig. 4. Execution of an application with Apache Flink

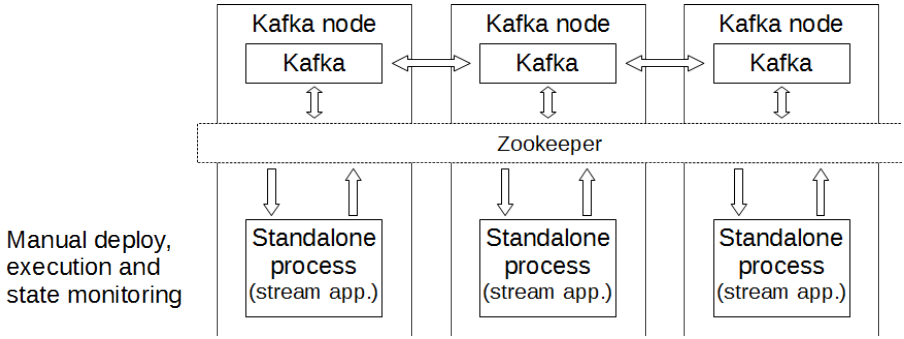

Рис. 5: Схема запуска приложения для Apache Kafka Streams
Fig. 5. Execution of an application with Apache Kafka Streams

4. Модели оценки характеристик потоковых фреймворков

Серьёзной проблемой выбора потоковых фреймворков в настоящее время является отсутствие единых и объективных критериев оценки производительности. Есть публикации, авторы которых проводят определённые сравнения, однако проблемой является узкая специализация и ограниченность применения этих тестов.

Одним из первых известных тестов производительности был так называемый линейный дорожный тест (the Linear Road benchmark) [20]. Тест имитирует дорожное движение в некотором абстрактном городе с параллельными дорогами. Каждый автомобиль раз в 30 секунд отправляет информацию о своём положении. Задачи потоковых фреймворков – определить текущий трафик на каждой из дорог, выявить аварии, учитывать количество активных полос движения, которое постоянно изменяется из-за аварий. Кроме того, требуется динамически подсчитывать затраты времени на проезд транспортными средствами по дорогам за заданный период времени. Проблемой применимости этого теста является то, что для каждого фреймворка необходима собственная реализация тестирующего приложения. Этот тест был разработан для оценки проекта Aurora и, несмотря на проработанность модели данных и критериев оценки, он так и не стал универсальным средством оценки производительности потоковых фреймворков.

В работе [31] рассматривается комбинированный тест StreamBench для оценки производительности и задержки обработки, в котором используются 7 программ тестирования с различной сложностью и различными операциями (фильтрация по образцу, поиск, извлечение поля, подсчёт количества слов и пр.). Проведено тестирование Apache Spark и Apache Storm. Этот тест также не получил развития, вероятно, потому, что исходные тексты не доступны.

В работе [25] представлен тест BigBench. Предполагается, что необходимо включать тестирование большого объема, различного типа данных и скорости обработки (volume, variety, velocity). Объем подразумевает петабайты данных,

различный тип – структурированные, слабо структурированные и неструктурированные данные, скорость обработки – требование "текучести данных" при решении задач ETL. Принципиальным отличием от предыдущих работ является то, что сделана попытка использования стандартизованных наборов тестов TPC-DS (TPC Benchmark DS: 'The' Benchmark Standard for decision support solutions including Big Data).

В работе [35] представлена модификация BigBench, в рамках которой сравниваются Apache Flink и Hive с использованием SQL-подобного языка HiveQL. Реализовано несколько запросов TPC-DS в терминах декларативной Java-модели Apache Flink и HiveQL. Широкого распространения эти тесты также не получили. В работе [39] представлено видение теста BigBench, как приближенного к решению конкретных бизнес-задач, включая потоковую обработку данных, машинного обучения, анализ графов, мультимедиа. Этот тест не доведён до реализации.

Обзор других тестов фреймворков больших данных, не ограниченный потоковыми фреймворками, приводится в работе [23].

Следует отметить, что несмотря на годы развития области потоковой обработки данных, сейчас нет ни готовых к использованию тестов, ни даже единых методик тестирования. Поэтому рассмотрим отдельные аспекты тестирования потоковых фреймворков.

Типовая схема тестирования приводится на рис. 6. Kafka(1) и Kafka(2) – входная и выходная очереди сообщений, которые в ряде случаев могут быть заменены самими потоковыми фреймворками.

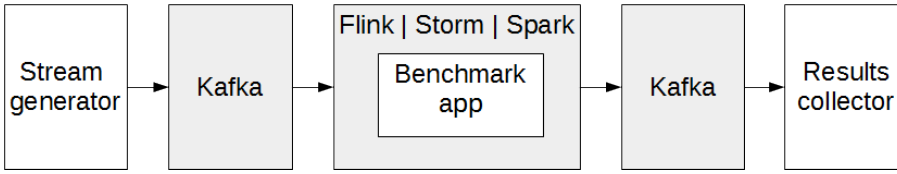

Рис. 6: Типовая схема тестирования потоковых фреймворков

Fig. 6 Typical benchmarking schema for streaming frameworks

4.1 Оценка задержки обработки данных

Одним из самых известных тестов потоковых фреймворков сейчас является Yahoo Streaming Benchmark [22]. Исходные тексты этого теста доступны и могут быть повторно использованы. Опубликованы результаты тестирования Apache Storm, Apache Spark Streaming и Apache Flink. Целью теста является оценка задержки времени обработки данных (Latency) при выполнении типовой цепочки обработки данных с простейшей агрегацией и чтением/записью данных в СУБД Redis. Целью теста не является оценка производительности, поэтому данный тест может быть использован лишь для

оценки применимости потоковых фреймворков для использования в интерактивных системах обработки данных.

В Yahoo Streaming Benchmark используется схема тестирования, приведённая на рис. 7. Задержка вычисляется по формуле:

$$window.final\_event\_latency = (window.last\_updated\_at - window.timestamp) - window.duration$$

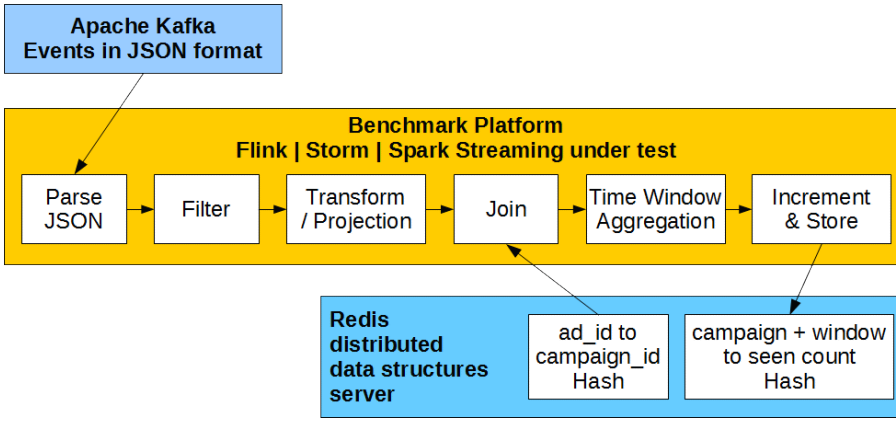

Рис. 7: Схема тестирования в Yahoo Streaming Benchmark

Fig. 7: Benchmarking schema in Yahoo Streaming Benchmark

В рамках теста реализовано 3 приложения для фреймворков Apache Storm, Apache Spark и Apache Flink.

4.2 Оценка автоматической подстройки конвейера обработки данных

Помимо задержки обработки, для интерактивных систем обработки данных важной характеристикой является регулярность выдачи результатов. Например, пусть установлена оконная операция с выдачей результата раз в 1 минуту. Независимо от того, сколько входных данных было получено, необходимо обеспечить обработку данных и выдачу результата. Подстройка скорости работы конвейера (backpressure) позволяет потоковому фреймворку обеспечить обработку данных в цепочке операторов со скоростью самого медленного оператора, исключая накопление перед ними больших массивов необработанных данных и потери возможности формирования отклика после них.

Модель оценки заключается в том, что на вход потоковой задачи, обрабатывающую данные в рамках скользящего окна, необходимо подать поток с плотностью больше, чем может быть выполнено за время, равное

смещению окна. То есть до момента переключения окна на следующий интервал надо обеспечить выдачу результата предыдущей обработки.

Пример, реализованный в https://github.com/rssdev10/spark-kafka-streaming, показывает, что в Apache Spark, например, отсутствуют эффективные механизмы подстройки конвейера. Это выражается в том, что если в момент фазы чтения данных из источника источник содержит больше данных, чем Spark может обработать, все данные будут загружены в Spark. Однако их полная обработка состоится неизвестно когда, независимо от установленного времени выдачи результата, см. рис. 8 b). Если же источник данных выдаёт входной поток с резкими всплесками, для Spark это приводит к резкому увеличению интервалов выдачи результатов. Например, вместо 10 секунд выдача результата может произойти раз в несколько минут. Если же рассматривать поведение Apache Flink в этой же ситуации, то результат обработки будет выдан почти гарантированно в установленное для окна время. См. рис. 8 a).

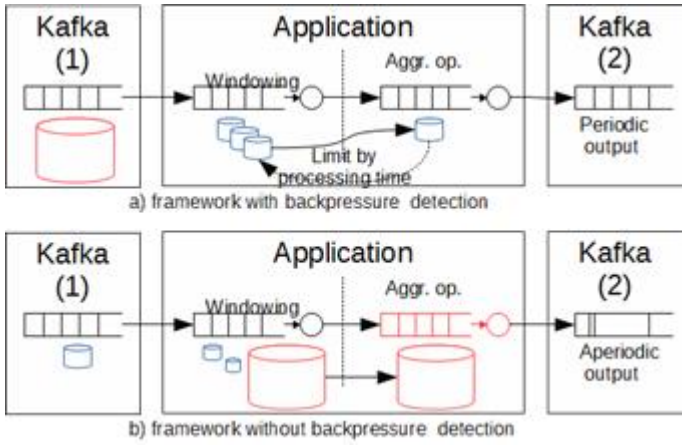

Рис. 8: Расположение данных в зависимости от подстройки конвейера

Fig. 8. Data concentration depending on presence of backpressure detector

Несмотря на то, что входные данные надо обработать в любом случае, тот факт, что фреймворк не успевает это сделать, означает лишь то, что данные где-то должны быть накоплены. Модель Flink упрощает дальнейшую обработку для программиста, которому не приходится делать дополнительные проверки времени выдачи результата.

То есть данная проверка позволяет оценить пригодность потоковых фреймворков для задач, требующих выдачи данных по строгому регламенту.

4.3 Оценка восстанавливаемости узлов кластера после сбоев

Распределённая кластерная система состоит из множества взаимодействующих узлов, каждый из которых может выйти из строя. Проверка потокового фреймворка на устойчивость работы в случае отключения отдельных узлов важна потому, что в отличии от пакетной обработки, потоковая задача запускается однократно и долговременно. Потоковая задача не может быть завершена без прекращения обработки запросов вообще. Если потоковый фреймворк не обеспечивает автоматическое восстановление узлов после сбоя, это ведёт к деградации производительности задачи. Именно поэтому в системах высокой доступности неприемлемо использование потоковых фреймворков, не способных обеспечить восстановление.

Модель оценки здесь достаточно простая. Она заключается в имитации сбоев узлов кластера, которая может быть реализована как вспомогательная программа, используемая совместно с любым другим тестом фреймворка. Необходимо имитировать аппаратные сбои (для потокового фреймворка выглядит как потеря процесса) или программные сбои задач (например, из-за нехватки оперативной памяти при выполнении).

Согласно тесту, проведенному в [33], например, Apache Spark 1.3 не способен восстановить работоспособность после одиночных сбоев. В то же время Apache Flink может работать долговременно без последствий от сбоев одиночных узлов.

4.4 Оценка возможности организации последовательной обработки данных без лишних сетевых обменов

В отличии от пакетной обработки, где данные могут быть заранее подготовлены и размещены в непосредственной близости от места обработки, в задачах потоковой обработки имеется проблема доставки этих данных. Кроме того, решаемые задачи в потоковой обработке часто алгоритмически более легковесны, чем задачи, решаемые при пакетной обработке. Это приводит к тому, что если фреймворк не способен обеспечить обработку данных (например, задачи агрегации по разделам) без лишних пересылок, то горизонтальное масштабирование производительности кластера будет невозможным.

Модель тестирования заключается в том, что данные для обработки должны быть предварительно распределены по узлам кластера, например, при помощи Apache Kafka, а потоковый фреймворк обязан построить план выполнения так, чтобы каждый узел использовал только данные из того раздела, который размещён на узле.

Проблема состоит в том, что в настоящее время потоковые фреймворки, как правило, лишь теоретически позволяют это сделать. Методы, позволяющие

организовать обработку подобным образом, обычно не документированы. К примеру, подобную схему можно организовать при помощи Apache Flink, но данные должны быть разложены по узлам при помощи нестандартной для Kafka функции распределения.

На рис. 9 представлена схема обработки с предварительным распределением данных. Цветом обозначены связанные операторы, распараллеливание которых обеспечивает потоковый фреймворк. Для простоты считаем, что входные данные одной и той же очереди Kafka(1) распределены на разделы по числу узлов кластера. Данные подвергаются некоторой предварительной обработке, операциям агрегации по каждому из разделов, после чего собираются и отправляются в выходные очереди Kafka(2).

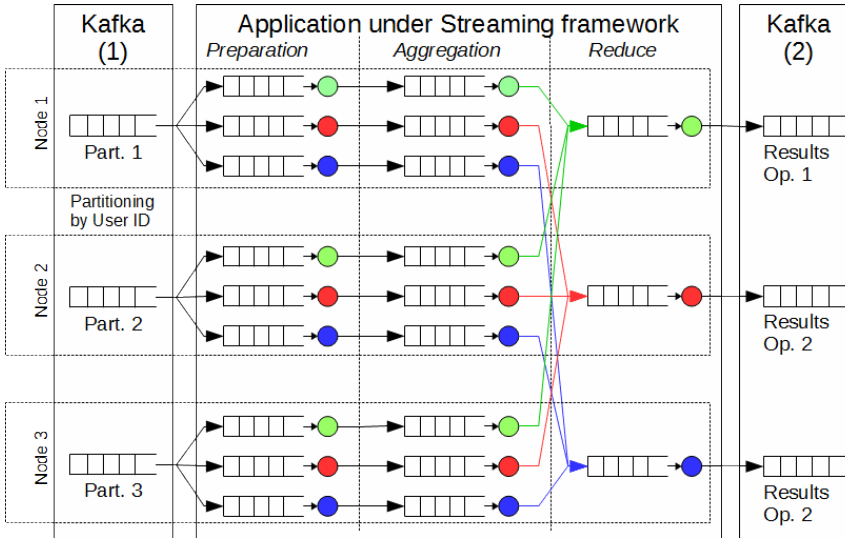

Рис. 9: Схема обработки с предварительным распределением данных

Fig. 9. Data processing with preliminary partitioning

4.5 Оценка предельных возможностей сохранения состояния (оконные операции)

Потоковые фреймворки используются для решения задач оценки входных данных по определённым критериям во времени, близком к реальному. Для этого применяются операции агрегации, которые могут быть выполнены над данными в рамках окна. При этом размер окна определяется решаемой задачей и может быть от нескольких секунд до суток. В последнем случае обычно используют так называемую лямбда-архитектуру [32], в рамках которой имеется постоянное хранилище для больших объёмов и долговременного хранения данных, а также небольшой объем оперативных

данных, принимаемых из входного потока. Однако надо понимать, что причина использования нескольких подсистем – ограничения существующих фреймворков.

Каппа-архитектура – это попытка построения систем обработки данных без использования отдельного долговременного хранилища и фреймворка для пакетной обработки данных. Любая операция агрегации предполагает накопление состояния, и, следовательно, наличие локального хранилища данных для каждого оператора, выполняемого в распределённой системе.

Легко подсчитать размер этого хранилища. Так, для случая входного потока с плотностью 100 тыс. сообщений в секунду и окна, размером в 1 час, необходимо хранить 360 млн. сообщений при условии, что потоковый фреймворк не создаёт дубликаты для случая скользящего окна с перекрытием. Или же данный размер следует кратно увеличить, если фреймворк реализует простейший случай скользящего окна.

Модель реализации этого теста достаточно проста. Необходимо обеспечить выполнение операций агрегации. Например, подходит решение задачи подсчёта количества уникальных пользователей и среднюю длительность сессии за 1 час с периодом обновления информации раз в 1 минуту на плотности запросов 100 тыс. в секунду.

Основная сложность этого теста заключается в том, что при подобных объемах поведение фреймворков становится нестабильным, и очень часто тесты вообще не могут быть выполнены.

4.6 Оценка прочих характеристик

В большинстве потоковых фреймворков в 2016-м году начала декларироваться поддержка языка SQL для написания запросов на выборку и обработку данных. Поскольку сейчас нет стандарта SQL, удовлетворяющего задачам потоковой обработки данных, разработчики потоковых фреймворков, по сути, ничем не ограничены в создании собственных диалектов SQL. Тем не менее, факт поддержки SQL заставляет задуматься над тем, как тестировать и этот аспект. В данный момент, вероятно, следует ставить вопрос о принципиальной возможности реализовать те или иные задачи с помощью SQL для каждого фреймворка конкретно и оценивать их производительность.

Кроме очевидных требований, приведённых выше, существуют специальные методы оптимизации выполнения графа операторов; состав этих методов описан в работе [27]. Среди них переупорядочивание операторов, устранение лишних операторов, балансировка нагрузки, рассылка состояния и пр. Очевидно, что это также влияет на скорость работы программ, однако тесты с учетом этих оптимизаций направлены в первую очередь на способность потоковых фреймворков автоматически обеспечивать оптимизацию не оптимальных программ. Это также влияет на скорость разработки и отладки программ.

Если вернуться назад к требованиям к потоковым системам реального времени, сформулированным в работе [40], то не рассмотренными остались требование 3 (дефектность входных потоков), поскольку это скорее задача прикладного программиста, а не фреймворка, а также требование 5 (интегрированность хранимых и потоковых данных), поскольку это зависит от модели информационной системы в целом.

5. Потоковые фреймворки с открытым исходным кодом

5.1 Apache Storm/Trident, Twitter Heron

Продукты Apache Storm [6] и Storm/Trident были популярны несколько лет назад и рекомендовались в качестве основы для систем потоковой обработки данных [32]. Однако в настоящее время их можно отнести к устаревшим программным продуктам, которые нельзя использовать в высоконагруженных системах. Основные недостатки – низкая производительность Trident, сложность настройки для работы в кластере, которая сводится к подбору параметров распараллеливания. Также следует отметить неспособность обеспечить нормированное время формирования ответа при резких всплесках плотности входного потока сообщений. Из отличительных достоинств следует назвать концепцию DRPC (distributed remote procedure call), в рамках которой Storm предоставляет программный интерфейс для интерактивных запросов, обеспечивая автоматическое перенаправление на свободные узлы кластера. Это также позволяет реализовывать микросервисы, способные выдать ответ на основе сохранённого состояния.

Twitter Heron [13] – новый проект, в котором декларируется обратная совместимость с приложениями, написанными для Apache Storm. Однако в данный момент продукт находится в разработке и не может быть рекомендован для реального использования.

5.2 Apache Spark Streaming

Apache Spark [5] является, пожалуй, самым известным фреймворком для решения задач машинного обучения. Модуль Apache Spark Streaming предназначен для потоковой обработки. Проблема Apache Spark заключается в его архитектуре. Изначально продукт ориентирован на пакетную обработку больших данных как замена Apache Hadoop и Map/Reduce. Apache Spark Streaming гипотетически позволяет решать потоковые задачи, однако наследие пакетной обработки сказывается негативно. Декларируемая возможность использования единого программного интерфейса для пакетного и потокового режимов работы, но в реальности это реализовано с большими ограничениями.

Рассмотрим схему тестирования фреймворка, представленную на рис. 10. Stream generator - генератор потока данных, Zookeeper - координатор кластера, Kafka - единственный узел Kafka, тестируемый фреймворк Flink или Spark, Benchmark – тестирующее приложение. Схема является простейшей, причём

вся генерация данных осуществляется на единственном узле управления, все остальные узлы - вычислительные. Это позволяет очень просто определить ограничения вычислительной сети.

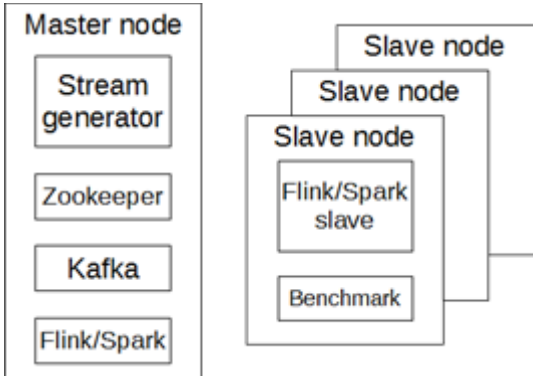

Рис. 10: Схема тестирования в режиме одного управляющего и множества вычислительных узлов

Fig. 10. Benchmarking with one master node and multiple slave nodes

На графике загрузки сети (рис. 11) очевиден характер работы Spark, где чётко прослеживаются фазы загрузки данных и фазы обработки. Это приводит к естественному физическому ограничению вычислительной сети и к ограничению производительности. Показатели загрузки сети получены при помощи утилиты dstat.

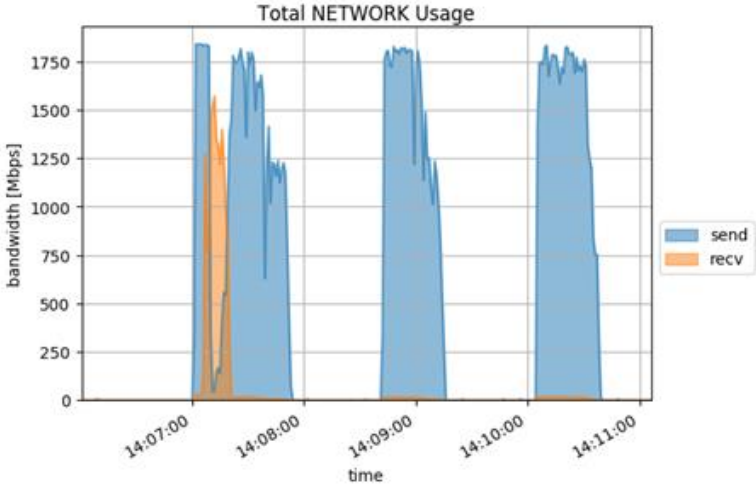

Рис. 11: График загрузки сетевого адаптера управляющего узла Spark

Fig. 11. Master node network utilization by Apache Spark

Существенной проблемой является то, что Spark не способен восстанавливать работу узлов кластера, которые выпали в результате сбоя [33]. Если речь идёт о пакетном режиме работы, это не является проблемой, поскольку пакетная задача будет выгружена после завершения обработки. В случае же потоков, где программы запускаются один раз и должны работать постоянно, это неприемлемо, поскольку ведёт к неминуемой деградации производительности.

Проблемой также является необходимость вручную подбирать и устанавливать ограничения плотности входного потока, поскольку особенность Spark в том, что не имея эффективных средств отслеживания обратного давления, всплеск входных данных приводит к тому, что Spark пытается захватить их все, игнорируя выставленное время обработки в окне.

Можно предположить, что в ближайшие 1-2 года разработчики не решатся изменить его ядро, поскольку потенциально это приведёт к потере совместимости с написанными ранее приложениями. В настоящее время продукт (версия 2.0.1) непригоден для работы в системах с нерегулярной нагрузкой, непригоден для интерактивных систем. Учитывая не слишком высокую производительность, высокое время задержки, а также массу ручных операций подбора параметров, можно предположить, что Spark будет одним из самых дорогих в эксплуатации продуктом.

5.3 Apache Flink

Apache Flink [2] позиционируется как универсальный фреймворк, то есть способный выполнять как потоковые, так и пакетные задачи, однако для того, чтобы избегать прямой конкуренции с Apache Spark, разработчики сместили основной акцент именно на обработку потоковых данных. Проект родился из академического проекта Stratosphere и отличается глубокой теоретической проработкой архитектуры [17].

В Apache Flink реализуется естественная потоковая обработка данных, гарантируется обработка сообщений строго одни раз, то есть "exactly-once", обеспечивается автоматическое балансирование нагрузки и восстановление после сбоев, а также присутствуют механизмы подстройки скорости обработки конвейера.

Из очевидных достоинств работы Apache Flink следует отметить непрерывность выполнения операторов без разделения на фазы загрузки данных и обработку. Тест, проведённый по схеме, представленной на рис. 10, показывает почти ровный график загрузки сетевого адаптера управляющего узла (см. рис. 12).

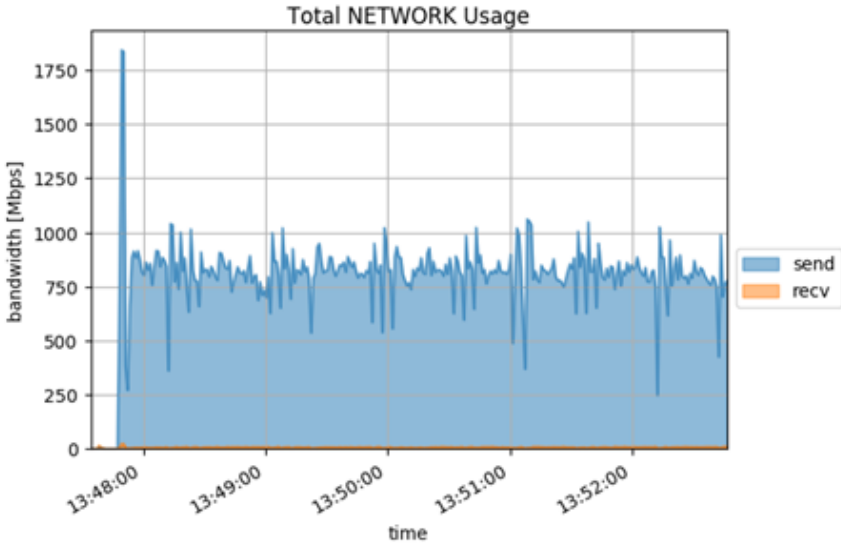

Рис. 12: График загрузки сетевого адаптера управляющего узла Flink

Fig. 12. Master node network utilization by Apache Flink

Apache Flink характеризуется низкими задержками выполнения [22] и высокой производительностью.

Из недостатков традиционно отмечают то, что проект достаточно молодой, и немногие крупные компании реально решили его использовать, однако динамика его использования за 2016 год говорит о достаточном его уровне развития.

Apache Flink имеет библиотеку для описания обработки в терминах CEP [42]]. Примером задачи CEP может быть выполнение действий при достижении некоторой температуры. Также поддерживается SQL [24].

5.4 Apache Kafka Streams

Kafka Streams [29] является результатом развития проекта Apache Kafka [3]. Поскольку Apache Kafka широко используется в бизнес-приложениях в качестве средства поддержки очередей сообщений, прочие потоковые фреймворки вынуждены обеспечивать чтение и отправку сообщений из Kafka. При этом на стыке безусловно возникают проблемы гарантированности обработки сообщений и производительности. Поэтому для компании Confluent было логичным предложить собственный инструмент потоковой обработки, который входит в состав Apache Kafka, начиная с версии 0.10. В настоящее время Kafka Streams – легковесная библиотека, позволяющая

выполнять традиционные операции с окнами, агрегацию данных и описывать собственные операторы для обработки данных.

Теоретическим плюсом данного фреймворка является доступность всех внутренних механизмов Apache Kafka, которые недоступны или не документированы для использования сторонними пользователями Kafka.

Разработчики Kafka делают акцент на том, что потоковые данные и таблицы по сути являются одним и тем же [29]. То есть таблица – это результат накопления некоторых событий за определенное время, или окно в терминах современного потокового процессора. Соответственно, в программном интерфейса Kafka предусмотрены классы KTable и KStream. Если же обеспечить долговременное хранение всего потока, получаем темпоральную СУБД.

Среди недостатков Kafka Streams следует отметить то, что продукт не является устойчивым, возможны изменения API, семантика операций агрегации отличается от других фреймворков. Например, операция aggregateByKey вместо накопления и выдачи нескольких значений в количестве, равном количеству ключей, будет выдавать частные агрегаты по каждому ключу на каждое входящее сообщение. Возможно, это поведение будет изменено в следующих версиях.

Ещё одним специфическим моментом Kafka Streams является то, что нет поддержки распараллеливания данных, масштабирования и автоматического контроля состояния, поскольку потоковое приложение для Kafka Streams – это автономное Java-приложение, связанное с Apache Kafka только стандартным протоколом взаимодействия. Иллюстрация работы приложения, использующего Kafka Streams, была приведена на рис. 5.

Если эти возможности не будут предоставлены программистам в ближайшее время, то прямым конкурентом Kafka Streams является Project Reactor [15]. Его достоинством является глубокая интеграция с Java (является кандидатом на включение в состав Java 9). Project Reactor уже использован в проекте Spring Framework 5. В активной разработке находится проект Reactor Kafka.

5.5 Apache Samza

Apache Samza [4] – потоковый фреймворк, демонстрирующий очень высокую скорость обработки сообщений и низкую задержку их обработки [41]. Samza позиционируется как продукт той же категории, что и Apache Storm.

Samza поддерживает режим обработки "exactly-once", то есть дубликаты сообщений возможны. Сохранение состояния обеспечивается за счёт БД «ключ-значение», связанной с каждой потоковой задачей.

Модель программирования – композиционная, причём программа создаётся в терминах классов Java, реализующих определённые интерфейсы. То есть, в отличии от Apache Flink или Spark, нет выбора языка программирования за пределами JVM.

Дополнительные особенности Samza рассматриваются в [14].

5.6 Apache Apex

Apache Apex [1] так же, как Apache Spark, Apache Flink, позиционируется как универсальный фреймворк для потоковой и пакетной обработки данных.

Так же, как и в Apache Flink, поддерживается режим "exactly-once", высокоуровневая декларативная модель программирования с возможностью работы с окнами данных и обработкой сообщений по их внутреннему времени или по системному времени. Поддерживаются средства ETL, имеется набор средств интеграции с различными очередями сообщений, включая Kafka, а также с различными СУБД. Декларируется интеграция со средствами Apache Beam, Apache SAMOA.

В статье [43] сравниваются Apache Apex и Apache Flink по схеме Yahoo Streaming Benchmark. При этом Apex показывает несколько меньшие абсолютные задержки и разброс их значений на больших объемах данных. Однако никаких оценок по другим критериям в статье нет.

6. Коммерческие программные продукты с закрытым исходным кодом

В отношении коммерческих потоковых фреймворков ситуация совершенно иная. В исследовании [36], например, сравниваются различные коммерческие фреймворки типа Cisco Connected Streaming Analytics, Data Torrent RTS, Esper Enterprise Edition, IBM Streams, Impetus Technologies StreamAnalytix, Oracle Stream Explorer, SAP Event Stream Processor, SAS Event Stream Processing, Software AG Apama Streaming Analytics Platform, SQLstream Blaze, Striim, TIBCO StreamBase, WSO2 Complex Event Processor и пр.

Главная проблема заключается в том, что часть вендоров выпустили продукты для потоковой обработки чисто номинально. В большинстве случаев потребитель не имеет возможности объективно сравнить их с открытыми потоковыми процессорами и должен довериться лишь авторитету марки. В данном случае следует отбрасывать компании, для которых программные разработки не являются основными. Указанный отчёт как раз и демонстрирует непроработанность продуктов у ряда компаний, что, скорее всего, приведёт в дальнейшем к отказу от их поддержки.

7. Заключение

В настоящее время наблюдается многообразие потоковых фреймворков. При этом следует отметить, с одной стороны, попытки разработчиков позиционировать свои продукты как универсальные и пригодные для всех случаев жизни, с другой стороны, существует тенденция создания специализированных фреймворков под конкретные задачи, свойственная

скорее крупным компаниям. И в одном, и в другом случае результат является набором компромиссов.

Дополнительно следует указать статью [11], где приводится сравнительная таблица 12-ти продуктов Apache, относящихся к группе потоковых фреймворков. В таблицу сведены лишь характеристики, декларируемые разработчиками этих продуктов.

Несмотря на годы развития потоковых фреймворков и их многообразие, до сих пор не существует никаких единых подходов для оценки их характеристик и экспериментальной проверки. Рассмотренные в этой статье методы могут быть использованы для проверки конкретных аспектов функционирования фреймворков и выбора подходящего продукта для конкретного случая. Задача создания общей методики тестирования, набора тестов и набора данных для проверки потоковых фреймворков всё ещё остаётся актуальной.

Список литературы

[1]. Apache Apex. https://apex.apache.org/. [Обращение 2017-01-02].

[2]. Apache Flink: Scalable Batch and Stream Data Processing. https://flink.apache.org/. [Обращение 2017-01-02].

[3]. Apache Kafka. https://kafka.apache.org/. [Обращение 2017-01-02].

[4]. Apache Samza. http://samza.apache.org/. [Обращение 2017-01-02].

[5]. Apache Spark™ - Lightning-Fast Cluster Computing. https://spark.apache.org/. [Обращение 2017-01-02].

[6]. Apache Storm. https://storm.apache.org/. [Обращение 2017-01-02].

[7]. Drools - Business Rules Management System (Java™, Open Source). https://www.drools.org/. [Обращение 2017-01-02].

[8]. Guaranteeing message processing. http://storm.apache.org/releases/current/Guaranteeing-message-processing.html. . [Обращение 2016-12-23].

[9]. RocksDB a persistent key-value store. http://rocksdb.org/. [Обращение 2017-01-02].

[10]. Spring. https://spring.io/. [Обращение 2017-01-02].

[11]. An Overview of Apache Streaming Technologies. https://databaseline.wordpress.com/2016/03/12/an-overview-of-apache-streaming-technologies/, 2016. [Обращение 2017-01-02].

[12]. Apache Flume. https://flume.apache.org/, 2016. [Обращение 2017-01-02].

[13]. Heron. A realtime, distributed, fault-tolerant stream processing engine from Twitter. https://twitter.github.io/heron/, 2016. [Обращение 2017-01-02].

[14]. Samza. Comparison Introduction. http://samza.apache.org/learn/documentation/latest/comparisons/introduction.html, 2016. [Обращение 2017-01-02].

[15]. Project Reactor. https://projectreactor.io/, 2017. [Обращение 2017-01-02].

[16]. Daniel J. Abadi, Don Carney, Ugur Çetintemel, Mitch Cherniack, Christian Convey, Sangdon Lee, Michael Stonebraker, Nesime Tatbul, and Stan Zdonik. Aurora: A new model and architecture for data stream management. The VLDB Journal, 12(2):120–139, August 2003.

[17]. Alexander Alexandrov, Rico Bergmann, Stephan Ewen, Johann-Christoph Freytag, Fabian Hueske, Arvid Heise, Odej Kao, Marcus Leich, Ulf Leser, Volker Markl, Felix

Naumann, Mathias Peters, Astrid Rheinländer, Matthias J. Sax, Sebastian Schelter, Mareike Höger, Kostas Tzoumas, and Daniel Warneke. The stratosphere platform for big data analytics. The VLDB Journal, 23(6):939–964, December 2014.

[18]. Alexander Alexandrov, Andreas Salzmann, Georgi Krastev, Asterios Katsifodimos, and Volker Markl. Emma in action: Declarative dataflows for scalable data analysis. In Fatma Özcan, Georgia Koutrika, and Sam Madden, editors, Proceedings of the 2016 International Conference on Management of Data, SIGMOD Conference 2016, San Francisco, CA, USA, June 26 - July 01, 2016, pages 2073–2076. ACM, 2016.

[19]. Henrique C. M. Andrade, Bugra Gedik, and Deepak S. Turaga. Fundamentals of Stream Processing: Application Design, Systems, and Analytics. Cambridge University Press, New York, NY, USA, 1st edition, 2014.

[20]. Arvind Arasu, Mitch Cherniack, Eduardo F. Galvez, David Maier, Anurag Maskey, Esther Ryvkina, Michael Stonebraker, and Richard Tibbetts. Linear road: A stream data management benchmark. In Mario A. Nascimento, M. Tamer Özsu, Donald Kossmann, Renée J. Miller, José A. Blakeley, and K. Bernhard Schiefer, editors, (e)Proceedings of the Thirtieth International Conference on Very Large Data Bases, Toronto, Canada, August 31 - September 3 2004, pages 480–491. Morgan Kaufmann, 2004.

[21]. Paris Carbone, Gyula Fóra, Stephan Ewen, Seif Haridi, and Kostas Tzoumas. Lightweight asynchronous snapshots for distributed dataflows. CoRR, abs/1506.08603, 2015.

[22]. S. Chintapalli, D. Dagit, B. Evans, R. Farivar, T. Graves, M. Holderbaugh, Z. Liu, K. Nusbaum, K. Patil, B. J. Peng, and P. Poulosky. Benchmarking streaming computation engines: Storm, flink and spark streaming. In 2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), pages 1789–1792, May 2016.

[23]. [Saliya Ekanayake. Towards a systematic study of big data performance and benchmarking. PhD thesis, the School of Informatics and Computing, Indiana University, United States – Indiana, 10 2016. http://pqdtopen.proquest.com/doc/1845860615.html?FMT=ABS.

[24]. Hueske Fabian. Stream Processing for Everyone with SQL and Apache Flink. https://flink.apache.org/news/2016/05/24/stream-sql.html, 2016. [Обращение 2017-01-02].

[25]. Ahmad Ghazal, Tilmann Rabl, Minqing Hu, Francois Raab, Meikel Poess, Alain Crolotte, and Hans-Arno Jacobsen. Bigbench: Towards an industry standard benchmark for big data analytics. In Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data, SIGMOD '13, pages 1197–1208, New York, NY, USA, 2013. ACM.

[26]. Lukasz Golab and M. Tamer Özsu. Issues in data stream management. SIGMOD Rec., 32(2):5–14, June 2003.

[27]. Martin Hirzel, Robert Soulé, Scott Schneider, Buğra Gedik, and Robert Grimm. A catalog of stream processing optimizations. ACM Comput. Surv., 46(4):46:1–46:34, March 2014.

[28]. Kreps Jay. Putting Apache Kafka To Use: A Practical Guide to Building a Stream Data Platform. https://www.confluent.io/blog/stream-data-platform-1/, https://www.confluent.io/blog/stream-data-platform-2/, 2015. [Обращение 2017-01-02].

[29]. Kreps Jay. Introducing kafka streams: Stream processing made simple - confluent. https://www.confluent.io/blog/introducing-kafka-streams-stream-processing-made-simple/ , 2016. [Обращение 2017-01-02].

[30]. Kostas, Ewen Stephan, and Metzger Robert. High-throughput, low-latency, and exactly-once stream processing with apache flink. http://data-artisans.com/high-throughput-low-latency-and-exactly-once-stream-processing-with-apache-flink/, 2015. [Обращение 2016-12-23].

[31]. Ruirui Lu, Gang Wu, Bin Xie, and Jingtong Hu. Stream bench: Towards benchmarking modern distributed stream computing frameworks. In Proceedings of the 2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing, UCC '14, pages 69–78, Washington, DC, USA, 2014. IEEE Computer Society.

[32]. Nathan Marz and James Warren. Big Data: Principles and Best Practices of Scalable Realtime Data Systems. Manning Publications Co., Greenwich, CT, USA, 1st edition, 2015.

[33]. Diana Matar. Benchmarking Fault-Tolerance in Stream Processing Systems. Master's thesis. TU-Berlin, 2016, 57 p.

[34]. Zaharia Matei, Wendell Patrick, and Das Tathagata. Diving into apache spark streaming's execution model. https://databricks.com/blog/2015/07/30/diving-into-apache-spark-streamings-execution-model.html, 2015. [Обращение 2016-12-23].

[35]. Guido Mazza. big data streaming processing engines under the umbrella of the apache foundation: benchmark and industrial applications. Master's thesis. Universita` degli Studi di Modena e Reggio Emilia, 2015. http://www.dbgroup.unimo.it/tesi/Magistrale/201516_Guido_Mazza_tesi.pdf

[36]. Gualtieri Mike, Curran Rowan, Kisker Holger, Miller Emily, and Izzi Matthew. The forrester wave™: Big data streaming analytics, q1 2016. http://www.cakesolutions.net/teamblogs/comparison-of-apache-stream-processing-frameworks-part-2, https://www.sas.com/content/dam/SAS/en_us/doc/analystreport/forrester-big-data-streaming-analytics-108218.pdf, 2016. [Обращение 2017-01-02].

[37]. Zapletal Petr. Comparison of apache stream processing frameworks: Part 1. http://www.cakesolutions.net/teamblogs/comparison-of-apache-stream-processing-frameworks-part-1, 2016. [Обращение 2016-12-23].

[38]. Zapletal Petr. Comparison of apache stream processing frameworks: Part 2. http://www.cakesolutions.net/teamblogs/comparison-of-apache-stream-processing-frameworks-part-2, 2016. [Обращение 2016-12-23].

[39]. Tilmann Rabl, Michael Frank, Manuel Danisch, Hans-Arno Jacobsen, and Bhaskar Gowda. The vision of bigbench 2.0. In Proceedings of the Fourth Workshop on Data Analytics in the Cloud, DanaC'15, pages 3:1–3:4, New York, NY, USA, 2015. ACM.

[40]. Michael Stonebraker, Uğur Çetintemel, and Stan Zdonik. The 8 requirements of real-time stream processing. SIGMOD Rec., 34(4):42–47, December 2005. Перевод http://citforum.ru/database/articles/stream_8_req/.

[41]. Feng Tao. Benchmarking Apache Samza: 1.2 million messages per second on a single node. https://engineering.linkedin.com/performance/benchmarking-apache-samza-12-million-messages-second-single-node, 2015. [Обращение 2016-12-01].

[42]. Rohrmann Till. Introducing Complex Event Processing (CEP) with Apache Flink. https://flink.apache.org/news/2016/04/06/cep-monitoring.html, 2016. [Обращение 2017-01-02].

[43]. Rozov Vlad. Throughput, Latency, and Yahoo! Performance Benchmarks. Is there a winner? https://community.mapr.com/community/exchange/blog/2016/12/05/throughput-latency-and-yahoo-performance-benchmarks-is-there-a-winner-by-vlad-rozov, 2016. [Обращение 2017-01-02].

Survey of streaming processing field

R.S. Samarev <samarev@acm.org>
Bauman Moscow State Technical University,
ul. Baumanskaya 2-ya, 5/1, Moscow, 105005, Russia

Abstract. This article is devoted to review of current state of streaming processing field including. This includes some historical aspects and mentioning of leaps of technologies development. Big part is attended to aspects of working of streaming processing frameworks at the point of view programmers which are using ones to create own streaming applications. These aspects includes framework selection issues which are criteria of selections, hidden aspects of functioning, existing benchmarks for big data and streaming frameworks review, and different benchmarking techniques description. Also, this article contains comprehensive list of references.

Keywords: big data, benchmark, apache storm, spark, flink, samza, apex, kafka

DOI: 10.15514/ISPRAS-2017-29(1)-13

For citation: Samarev R.S., Review of streaming processing field. *Trudy ISP RAN/Proc. ISP RAS*, vol. 29, issue 1, 2017. pp. 231-260 (in Russian). DOI: 10.15514/ISPRAS-2017-29(1)-13

References

[1]. Apache Apex. https://apex.apache.org/. [Online; accessed 2017-01-02].
[2]. Apache Flink: Scalable Batch and Stream Data Processing. https://flink.apache.org/. [Online; accessed 2017-01-02].
[3]. Apache Kafka. https://kafka.apache.org/. [Online; accessed 2017-01-02].
[4]. Apache Samza. http://samza.apache.org/. [Online; accessed 2017-01-02].
[5]. Apache Spark™ - Lightning-Fast Cluster Computing. https://spark.apache.org/. [Online; accessed 2017-01-02].
[6]. Apache Storm. https://storm.apache.org/. [Online; accessed 2017-01-02].
[7]. Drools - Business Rules Management System (Java™, Open Source). https://www.drools.org/. [Online; accessed 2017-01-02].
[8]. Guaranteeing message processing. http://storm.apache.org/releases/current/Guaranteeing-message-processing.html. [Online; accessed 2016-12-23].
[9]. RocksDB a persistent key-value store. http://rocksdb.org/. [Online; accessed 2017-01-02].
[10]. Spring. https://spring.io/. [Online; accessed 2017-01-02].
[11]. An Overview of Apache Streaming Technologies. https://databaseline.wordpress.com/2016/03/12/an-overview-of-apache-streaming-technologies/, 2016. [Online; accessed 2017-01-02].
[12]. Apache Flume. https://flume.apache.org/, 2016. [Online; accessed 2017-01-02].
[13]. Heron. A realtime, distributed, fault-tolerant stream processing engine from Twitter. https://twitter.github.io/heron/, 2016. [Online; accessed 2017-01-02].

[14]. Samza. Comparison Introduction. http://samza.apache.org/learn/documentation/latest/comparisons/introduction.html, 2016. [Online; accessed 2017-01-02].
[15]. Project Reactor. https://projectreactor.io/, 2017. [Online; accessed 2017-01-02].
[16]. Daniel J. Abadi, Don Carney, Ugur Çetintemel, Mitch Cherniack, Christian Convey, Sangdon Lee, Michael Stonebraker, Nesime Tatbul, and Stan Zdonik. Aurora: A new model and architecture for data stream management. The VLDB Journal, 12(2):120–139, August 2003.
[17]. Alexander Alexandrov, Rico Bergmann, Stephan Ewen, Johann-Christoph Freytag, Fabian Hueske, Arvid Heise, Odej Kao, Marcus Leich, Ulf Leser, Volker Markl, Felix Naumann, Mathias Peters, Astrid Rheinländer, Matthias J. Sax, Sebastian Schelter, Mareike Höger, Kostas Tzoumas, and Daniel Warneke. The stratosphere platform for big data analytics. The VLDB Journal, 23(6):939–964, December 2014.
[18]. Alexander Alexandrov, Andreas Salzmann, Georgi Krastev, Asterios Katsifodimos, and Volker Markl. Emma in action: Declarative dataflows for scalable data analysis. In Fatma Özcan, Georgia Koutrika, and Sam Madden, editors, Proceedings of the 2016 International Conference on Management of Data, SIGMOD Conference 2016, San Francisco, CA, USA, June 26 - July 01, 2016, pages 2073–2076. ACM, 2016.
[19]. Henrique C. M. Andrade, Bugra Gedik, and Deepak S. Turaga. Fundamentals of Stream Processing: Application Design, Systems, and Analytics. Cambridge University Press, New York, NY, USA, 1st edition, 2014.
[20]. Arvind Arasu, Mitch Cherniack, Eduardo F. Galvez, David Maier, Anurag Maskey, Esther Ryvkina, Michael Stonebraker, and Richard Tibbetts. Linear road: A stream data management benchmark. In Mario A. Nascimento, M. Tamer Özsu, Donald Kossmann, Renée J. Miller, José A. Blakeley, and K. Bernhard Schiefer, editors, (e)Proceedings of the Thirtieth International Conference on Very Large Data Bases, Toronto, Canada, August 31 - September 3 2004, pages 480–491. Morgan Kaufmann, 2004.
[21]. Paris Carbone, Gyula Fóra, Stephan Ewen, Seif Haridi, and Kostas Tzoumas. Lightweight asynchronous snapshots for distributed dataflows. CoRR, abs/1506.08603, 2015.
[22]. S. Chintapalli, D. Dagit, B. Evans, R. Farivar, T. Graves, M. Holderbaugh, Z. Liu, K. Nusbaum, K. Patil, B. J. Peng, and P. Poulosky. Benchmarking streaming computation engines: Storm, flink and spark streaming. In 2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), pages 1789–1792, May 2016.
[23]. Saliya Ekanayake. Towards a systematic study of big data performance and benchmarking. PhD thesis, the School of Informatics and Computing, Indiana University, United States – Indiana, 10 2016. http://pqdtopen.proquest.com/doc/1845860615.html?FMT=ABS.
[24]. Hueske Fabian. Stream Processing for Everyone with SQL and Apache Flink. https://flink.apache.org/news/2016/05/24/stream-sql.html, 2016. [Online; accessed 2017-01-02].
[25]. Ahmad Ghazal, Tilmann Rabl, Minqing Hu, Francois Raab, Meikel Poess, Alain Crolotte, and Hans-Arno Jacobsen. Bigbench: Towards an industry standard benchmark for big data analytics. In Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data, SIGMOD '13, pages 1197–1208, New York, NY, USA, 2013. ACM.
[26]. Lukasz Golab and M. Tamer Özsu. Issues in data stream management. SIGMOD Rec., 32(2):5–14, June 2003.

[27]. Martin Hirzel, Robert Soulé, Scott Schneider, Buğra Gedik, and Robert Grimm. A catalog of stream processing optimizations. ACM Comput. Surv., 46(4):46:1–46:34, March 2014.

[28]. Kreps Jay. Putting Apache Kafka To Use: A Practical Guide to Building a Stream Data Platform. https://www.confluent.io/blog/stream-data-platform-1/, https://www.confluent.io/blog/stream-data-platform-2/, 2015. [Online; accessed 2017-01-02].

[29]. Kreps Jay. Introducing kafka streams: Stream processing made simple - confluent. https://www.confluent.io/blog/introducing-kafka-streams-stream-processing-made-simple/ , 2016. [Online; accessed 2017-01-02].

[30]. Tzoumas Kostas, Ewen Stephan, and Metzger Robert. High-throughput, low-latency, and exactly-once stream processing with apache flink. http://data-artisans.com/high-throughput-low-latency-and-exactly-once-stream-processing-with-apache-flink/, 2015. [Online; accessed 2016-12-23].

[31]. Ruirui Lu, Gang Wu, Bin Xie, and Jingtong Hu. Stream bench: Towards benchmarking modern distributed stream computing frameworks. In Proceedings of the 2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing, UCC '14, pages 69–78, Washington, DC, USA, 2014. IEEE Computer Society.

[32]. Nathan Marz and James Warren. Big Data: Principles and Best Practices of Scalable Realtime Data Systems. Manning Publications Co., Greenwich, CT, USA, 1st edition, 2015.

[33]. Diana Matar. Benchmarking Fault-Tolerance in Stream Processing Systems. Master's thesis. TU-Berlin, 2016, 57 pp.

[34]. Zaharia Matei, Wendell Patrick, and Das Tathagata. Diving into apache spark streaming's execution model. https://databricks.com/blog/2015/07/30/diving-into-apache-spark-streamings-execution-model.html, 2015. [Online; accessed 2016-12-23].

[35]. Guido Mazza. big data streaming processing engines under the umbrella of the apache foundation: benchmark and industrial applications. Master's thesis. Universita` degli Studi di Modena e Reggio Emilia, 2015. http://www.dbgroup.unimo.it/tesi/Magistrale/201516_Guido_Mazza_tesi.pdf

[36]. Gualtieri Mike, Curran Rowan, Kisker Holger, Miller Emily, and Izzi Matthew. The forrester wave™: Big data streaming analytics, q1 2016. http://www.cakesolutions.net/teamblogs/comparison-of-apache-stream-processing-frameworks-part-2, https://www.sas.com/content/dam/SAS/en_us/doc/analystreport/forrester-big-data-streaming-analytics-108218.pdf, 2016. [Online; accessed 2017-01-02].

[37]. Zapletal Petr. Comparison of apache stream processing frameworks: Part 1. http://www.cakesolutions.net/teamblogs/comparison-of-apache-stream-processing-frameworks-part-1, 2016. [Online; accessed 2016-12-23].

[38]. Zapletal Petr. Comparison of apache stream processing frameworks: Part 2. http://www.cakesolutions.net/teamblogs/comparison-of-apache-stream-processing-frameworks-part-2, 2016. [Online; accessed 2016-12-23].

[39]. Tilmann Rabl, Michael Frank, Manuel Danisch, Hans-Arno Jacobsen, and Bhaskar Gowda. The vision of bigbench 2.0. In Proceedings of the Fourth Workshop on Data Analytics in the Cloud, DanaC'15, pages 3:1–3:4, New York, NY, USA, 2015. ACM.

[40]. Michael Stonebraker, Uğur Çetintemel, and Stan Zdonik. The 8 requirements of real-time stream processing. SIGMOD Rec., 34(4):42–47, December 2005.

[41]. Feng Tao. Benchmarking Apache Samza: 1.2 million messages per second on a single node. https://engineering.linkedin.com/performance/benchmarking-apache-samza-12-million-messages-second-single-node, 2015. [Online; accessed 2016-12-01].

[42]. Rohrmann Till. Introducing Complex Event Processing (CEP) with Apache Flink. https://flink.apache.org/news/2016/04/06/cep-monitoring.html, 2016. [Online; accessed 2017-01-02].

[43]. Rozov Vlad. Throughput, Latency, and Yahoo! Performance Benchmarks. Is there a winner? https://community.mapr.com/community/exchange/blog/2016/12/05/throughput-latency-and-yahoo-performance-benchmarks-is-there-a-winner-by-vlad-rozov, 2016. [Online; accessed 2017-01-02].