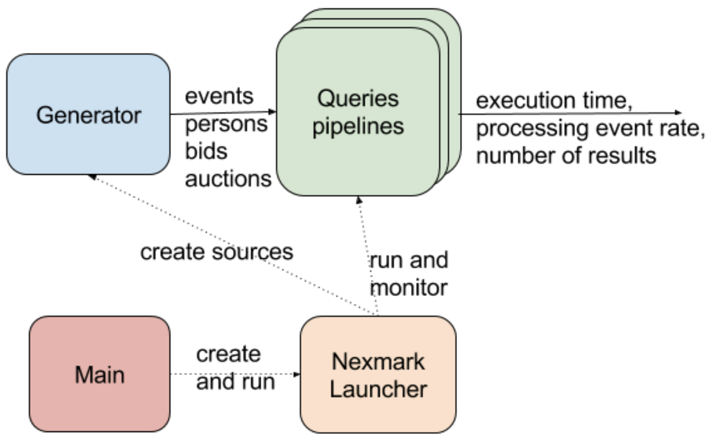
Quick presentation of Nexmark Code

Components of NexMark



- Generator:
  - generation of timestamped events (bids, persons, auctions) correlated between each other
- NexmarkLauncher:
  - creates sources that use the generator
  - queries pipelines launching, monitoring
- Output metrics:
  - Each query includes ParDos to update metrics
  - execution time, processing event rate, number of results, but also invalid auctions/bids, …

- Modes:
    - Batch mode: test data is finite and uses a BoundedSource
    - Streaming mode: test data is finite but uses an UnboundedSource to trigger streaming mode in runners

Queries pseudo code

Query 0 (not part of original NexMark): Pass-through.

- Allows us to measure the monitoring overhead.
    - serializes and deserializes using coder
    - Uses Aggregator for byte size counter

Query 1: What are the bid values in Euro's?

- Simple map
    - Filter + ParDo to extract bids out of events
    - ParDo that outputs Bid objects with price converted

Query 2: Find bids with specific auction ids and show their bid price.

- Illustrates simple filter
    - Filter + ParDo to **extract** bids out of events
    - Filter to **keep** bids with correct auctionId
    - ParDo that **outputs** AuctionPrice(auction, price) objects

Query 3: Who is selling in particular US states?

- Illustrates incremental join of the auctions and the persons collections
- uses global window and using per-key state and timer APIs
    - Apply global window to events with trigger repeatedly after at least nbEvents in pane => results will be materialized each time nbEvents are received.
    - **input1**: collection of auctions events **filtered** by category and **keyed** by seller id
    - **input2**: collection of persons events **filtered** by US state codes and **keyed** by person id
    - CoGroupByKey to **group** auctions and persons by personId/sellerId + tags to distinguish persons and auctions
    - ParDo to do the **incremental join**: auctions and person events can arrive out of order
        - person element **stored** in persistent state in order to match future auctions by that person. Set a timer to **clear** the person state after a TTL
        - auction elements **stored** in persistent state until we have seen the corresponding person record. Then, it can be **output** and **cleared**
    - **output** NameCityStateId(person.name, person.city, person.state, auction.id) objects

Query 4: What is the average selling price for each auction category?

- Illustrates sliding windows and aggregation
    - Apply Wining-bids
    - ParDo to key winning-bids by category
    - apply sliding windows to have a period of time
    - apply Mean.perKey (key = category)
    - ParDo that outputs CategoryPrice(categoryId, avgPrice)
    -

Query 5: Which auctions have seen the most bids in the last period?

- Illustrates sliding windows and combiners (i.e. reducers) to compare the elements in auctions Collection
    - **Input**: (sliding) window (to have a **result over** 1h period **updated** every 1 min) collection of bids events
    - ParDo to **replace** bid elements by their auction id
    - Count.PerElement to **count the occurrences** of each auction id
    - Combine.globally to **select** only the auctions with the **maximum number** of bids
        - BinaryCombineFn to **compare** one to one the elements of the collection (auction id occurrences, i.e. number of bids)
        - Return KV(auction id, max occurrences)
    - **output:** *AuctionCount(auction id, max occurrences)* objects

Query 6: What is the average selling price per seller for their last 10 closed auctions?

- Illustrates specialized combiner
    - Apply winning-bids
    - ParDo to **key** the winning-bids by sellerId
    - apply GlobalWindow + trigerring at **each** element (to have a continuous flow of **updates at each** new winning-bid)
    - Combine.perKey to calculate **average** of last 10 winning bids for each seller. Need specialized CombineFn because of 10 closed auctions
        - **create** Arraylist accumulators for chunks of data
        - **add all** elements of the chunks to the accumulators, **sort** them by bid timeStamp then price **keeping last 10** elements
        - iteratively **merge** the accumulators until there is only one: just **add** all bids of all accumulators to a final accumulator and **sort** by timeStamp then price **keeping last** 10 elements
        - extractOutput: **sum** all the prices of the bids and **divide** by accumulator size
    - ParDo that **outputs** SellerPrice(sellerId, avgPrice)

Query 7: What are the highest bids per period?

- Could have been implemented with a combiner like query5 but deliberately implemented using Max(prices) as a side input and illustrate fanout.
- Fanout is a redistribution using an intermediate implicit combine step to reduce the load in the final step of the Max transform
    - **input**: (fixed) windowed collection of bids events
    - ParDo to **replace** bids by their price
    - Max.withFanout to get the **max per window** and use it as a side input for next step. Fanout is useful if there are many events to be computed in a window using the Max transform.
    - ParDo on the bids with side input to **output** the bid if bid.price equals maxPrice (that comes from side input)
    -

Query 8: Who has entered the system and created an auction in the last period?

- Illustrates simple join
    - Filter + ParDo to **extract** persons out of events
    - Apply fixed windows to have a **period**
    - ParDo to **key** collection by personId
    - Filter + ParDo to **extract** auctions out of events
    - Apply fixed windows to have a **period**
    - ParDo to **key** collection by sellerId
    - CoGroupByKey to **group** persons and auctions by personId/sellerId + tag persons and auctions
    - ParDo to **output** IdNameReserve(person.id, person.name, auction.reserve) for each auction

Query 9 Winning-bids (not part of original NexMark): extract the most recent of the highest bids

- Illustrates custom window function to reconcile auctions and bids + join them
  - **input**: collection of events
  - Apply custom windowing function to **temporarily reconcile** auctions and bids events in the same custom window (AuctionOrBidWindow)
    - **assign** auctions to window [auction.timestamp, auction.expiring]
    - **assign** bids to window [bid.timestamp, bid.timestamp + expectedAuctionDuration (generator configuration parameter)]
    - **merge** all 'bid' windows into their corresponding 'auction' window, provided the auction has not expired.
  - Filter + ParDos to **extract** auctions out of events and **key** them by auction id
  - Filter + ParDos to **extract** bids out of events and **key** them by auction id
  - CogroupByKey (groups values of PCollections<KV> that share the same key) to **group** auctions and bids by auction id + tags to distinguish auctions and bids
  - ParDo to
    - **determine best** bid price: verification of **valid** bid, **sort** prices by **price ASC** then **time DESC** and **keep the max price**
    - and **output** AuctionBid(auction, bestBid) objects

Query 10 (not part of original NexMark):Log all events to GCS groupByfiles

- windows with large side effects on firing
  - ParDo to key events by their shardId (number of shards is a config item)
  - Apply fixed windows with composite triggering that fires when each sub-triger (executed in order) fires
    - repeatedly
      - after at least maxLogEvents in pane
      - or finally when watermark pass the end of window
    - Repeatedly
      - after at least maxLogEvents in pane
      - or processing time pass the first element in pane + lateDelay
    - With allowedLateness of 1 day (so that any late date will stall the pipeline and be noticeable)
  - GroupByKey to group events by shardId
  - ParDo to construct the outputStreams (fileNames contain shardId) and encode each event to that outputStream + form pairs with key = null key and value = outputFile (represents a fileName with various added information)
  - apply fixed window with default trigger and lateness of 1 day to clear complex triggerring
  - GroupByKey all outputFiles together (they have the same key) to have one file per window
  - ParDo to write all the lines to files in Google Cloud Storage

Query 11 (not part of original NexMark): How many bids did a user make in each session he was active?

- Illustrates session windows + triggering on the bids collection
    - **input**: collection of bids events
    - ParDo to **replace** bids with their bidder id
    - Apply session windows with gap duration = windowDuration (configuration item) and trigger repeatedly after at least nbEvents in pane => each window (i.e. session) will contain bid ids received since last windowDuration period of inactivity and materialized every nbEvents bids
    - Count.perElement to **count** bids per bidder id (number of occurrences of bidder id)
    - **output** idsPerSession(bidder, bidsCount) objects

Query 12 (not part of original NexMark): How many bids does a user make within a fixed processing time limit?

- Illustrates working in processing time in the Global window to count occurrences of bidder
  - **input**: collection of bid events
  - ParDo to **replace** bids by their bidder id
  - Apply global window with trigger repeatedly after processingTime pass the **first** element in pane + windowDuration (configuration item) => each pane will contain elements processed within windowDuration time
  - Count.perElement to **count** bids per bidder id (occurrences of bidder id)
  - **output** BidsPerWindow(bidder, bidsCount) objects