[YUNIKORN-42]
High efficient scheduling events framework phase 1

Use cases:

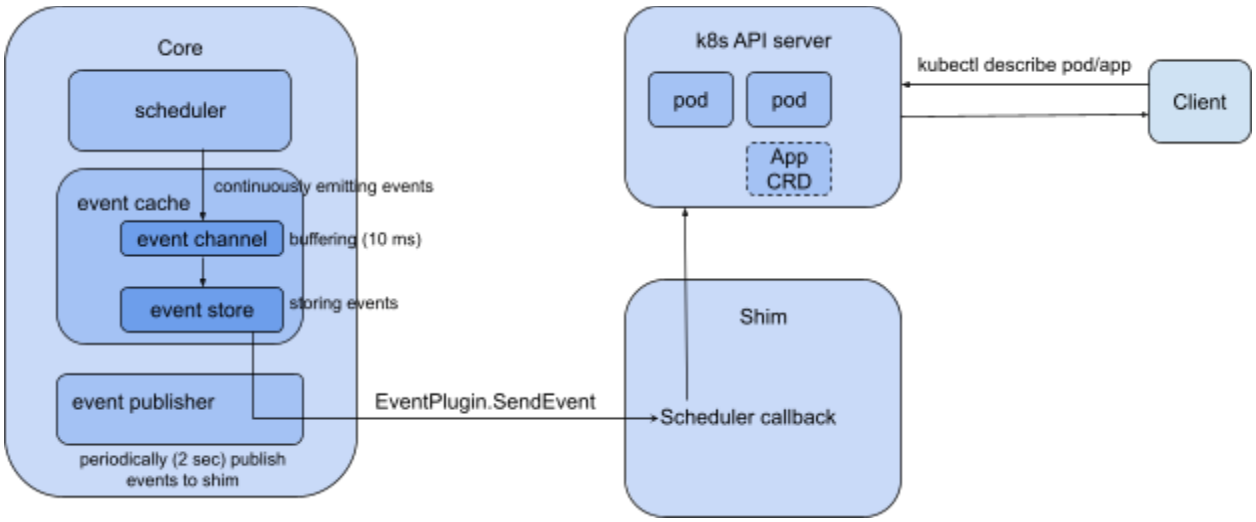
- Primary use case: Troubleshooting pending pods, why we cannot allocate it.
- Be able to look at queue/application/node/pod level and see: why we cannot allocate pod X on node Y.

Requirements:

- Events can be retrieved from K8s events (like kubectl describe pod). (P0)
- Avoid too much impact on performance. (P1)
- Events and diagnostics can be retrieved from YuniKorn UI / REST API. (P2)
- Be able to filter events/diagnostics based on queue/app/node/pod. (P2)

Implementation

Architecture



- **EventCache**: event cache is a cache to store object events, for different types of objects such as request, app, node etc. It maintains some eventStore to store these event records.
    - Uses an EventChannel to buffer the continuous flow of events generated by the scheduler.
    - Merges the events to the EventStore in every 10ms.
- **EventStore**: stores events for a certain type of object. This is an interface that can have different implementations. The very basic one is a fixed size store, which caches a certain number of events per object (e.g per request).
    - The store deletes the events upon retrieve.
- **EventPublisher**: the publisher retrieves events from the eventCache and determines how to publish these events. It leverages the scheduler plugin to publish events to the shim side. Can have different/multiple implementations in the future.
    - Periodically sends the event as EventRecords through the Scheduler callback interface. More precisely, EventPlugin's SendEvent function is called with the EventRecords.
- In the Shim side a Scheduler callback is invoked that directly pushes these events to K8s.
- The client can get these events through *kubectl describe* command.

Update of protocols

Update in SI

```
message EventRecord {
   enum Type {
      REQUEST = 0;
      APP = 1;
      NODE = 2;
      QUEUE = 3;
   }

   // the type of the object associated with the event
   Type type = 1;
   // ID of the object associated with the event
   string objectID = 2;
   // the group this object belongs to
   // it specifies the application ID for allocations and the queue for
applications
   string groupID = 3;
   // the reason of this event
   string reason = 4;
   // the detailed message as string
```

```
   string message = 5;
   // timestamp of the event
   int64 timestampNano = 6;
}
```

Update of SchedulerPlugins in Core

```
type EventPlugin interface {
      // This plugin is responsible for transmitting events to the shim
      // Events can be further exposed from the shim.
      SendEvent(events []*si.EventRecord) error
}
```