Splitting large Build Event Protocol events

Author: *buchgr@google.com, lpino@google.com*
Approvers: *aehlig@google.com, kxu@google.com, eduardocolaco@google.com*
Last Updated: *2018-08-24*[1]
Status: *Draft,* ***In review****, Approved, Implemented*
Tracking bug(s): #5715

This document is world commentable.

Objective

Provide a general approach for splitting the list of child events of large BEP events. Initially we'll apply this design to **only** split *PatternExpanded* but we plan to use as a general *example* on how to split large events.

Background

The *PatternExpanded* build event has one child event for every expanded target. Child events are announced by embedding their `BuildEventId` in a repeated field in the parent event. For builds with tens of thousands of targets or more the parent event can thus get very large in size (tens of megabytes) which can be a problem for some consumers of the Build Event Protocol (BEP). Most notably the Build Event Service (BES) protocol, the network transport protocol for the BEP, expects build events to be at most 50 MB in size and the gRPC library's default max message size is even smaller at 4 MB. Further, the protocol buffer documentation recommends individual messages to be no larger than 1MB in size.

Detailed Design

We propose to introduce a new build event *ChildEventsContinuation* whose sole purpose is to announce child events that didn't fit in a parent event. The first *ChildEventsContinuation* event will be announced as a child of e.g. the *PatternExpanded* event or a child of another *ChildEventsContinuation*. The subsequent example should make things clear.

Example

The *PatternExpanded* event below announces 3000 child events.

```
PatternExpanded {
  id : patternExpandedId { pattern : "..."},
  children : [targetConfiguredId {label: ":1"},
```

[1] Date of creation: 2018-07-26

```
                targetConfiguredId {label: ":2"}, ...,
                targetConfiguredId {label: ":3000"}
}
```

We want to split it up into multiple events using *ChildEventsContinuation* continuation events that contain at most 1000 child events each.

```
PatternExpanded {
  id : patternExpandedId { pattern : "..."},
  children : [targetConfiguredId {label: ":1"}, ...,
              targetConfiguredId {label: ":1000"},
              childEventsContinuationId { id : "unique1"}]
}

ChildEventsContinuation {
  id : childEventsContinuationId { id : "unique1" },
  logical_parent_id : patternExpandedId { pattern : "..."},
  children : [targetConfiguredId {label: ":1001"},
              ..., targetConfiguredId {label: ":2000"},
              childEventsContinuationId { id : "unique2"}]
}

ChildEventsContinuation {
  id : childEventsContinuationId { id : "unique2" },
  logical_parent_id : patternExpandedId { pattern : "..."},
  children : [targetConfiguredId {label: ":2001"}, ...,
              targetConfiguredId {label: ":3000"}]
}
```

The changes to the BEP protobuf definition are summarized below

```
message BuildEventId {
  ...

  message ChildEventsContinuationId {
    string id = 1;
  }

  oneof id {
    ...

    ChildEventsContinuation child_events_continuation = 23;
  }
}

message ChildEventsContinuation {
}

message BuildEvent {
  ...

  oneof payload {
    ...

    ChildEventsContinuation child_events_continuation = 25;
  }
}
```

Code Changes in Bazel

We propose to merge the `ChainableEvent` interface into the *BuildEvent* interface and to further change the method signature of *getChildrenEvents()* from

```
Collection<BuildEventId> getChildrenEvents()
```

to

```
List<BuildEventOrId> getChildrenEvents()
```

As the name suggests the `BuildEventOrId` type provides either a *BuildEvent* or a *BuildEventId*. That is a *BuildEvent* has now the ability to split itself up into multiple *BuildEvents* for serialization purposes. This interface should be general and powerful enough to in the future also allow other build events (i.e. *OptionsParsed*) to split themselves up to without code changes outside the event implementation.

Note that the collection type was changed to *j.u.List* because the order of the embedded child events matters now. That is events announcing child events need to appear before the child events in the list.

```
public interface BuildEvent extends ExtendedEventHandler.Postable {
  ...
  /**
   * Provides either a {@link BuildEventId} or a {@link BuildEvent}.
   */
  final class BuildEventOrId {

    private final BuildEventId id;
    private final BuildEvent event;

    public BuildEventOrId(BuildEventId id) {
      this.id = Preconditions.checkNotNull(id);
      this.event = null;
    }

    public BuildEventOrId(BuildEvent event) {
      this.event = Preconditions.checkNotNull(event);
      this.id = null;
    }
    @Nullable
    public BuildEventId id() {
      return id;
    }

    @Nullable
    public BuildEvent event() {
      return event;
    }
  }

  /**
   * Provide the children of the event.
   *
   * <p>It is a requirement of a well-formed event stream that for every
   * event that does not indicate the beginning of a new build, at least
   * one parent be present before the event itself.
   * However, more parents might appear later in the stream (e.g.,
   * if a test suite expanded later discovers that a test that is already
   * completed belongs to it).
   *
   * <p>A build-event stream is finished if and only if all
   * announced children have occurred.
   */
  List<BuildEventOrId> getChildrenEvents();

  ...

  /**
   * Provide a binary representation of the event.
   *
   * <p>Provide a presentation of the event according to the specified
  * binary format, as appropriate protocol buffer.
  *
  * <p>The serialized size must not exceed 1 MiB.
  */
  BuildEventStreamProtos.BuildEvent asStreamProto(
    BuildEventContext context);
}
```

Furthermore the changes in the `BuildEventStreamer` should be minimally invasive. After having posted a *BuildEvent* we need to recurse through the embedded child events and post them in the order they appear. If there are concerns of stack overflows we may choose to implement the recursion iteratively.

```
@Subscribe
public void buildEvent(BuildEvent event) {
  ...

  post(event);
```

```
    ...

    for (BuildEventOrId childEvent : event.getChildrenEvents()) {
      if (childEvent.event() != null) {
        buildEvent(childEvent.event());
      }
    }
    ...
  }
```

Migration Path

The proposed changes are not backwards compatible and might break clients who don't support *ChildEventsContinuation events.* We propose to add a flag --*build_event_splitting* which allows to disable *ChildEventsContinuation*. We'll roll out in three phases: opt-in (the flag exists), opt-out (the flag is the default) and mandatory (the flag is a no-op).