Ad hoc Metrics Based on Generic Trace Collection

wangxianzhu@chromium.org

This document is public.

Note that “ad hoc metrics” here has nothing to do with “ad hoc measurements” in telemetry documentation.

Use Cases

Sometimes we want to measure some ad hoc metrics. We want this to capture metrics similar to what real-world users would encounter, but can accept running the analysis on initial page loads of the top 10,000 pages. We need the result to be available quickly, and often run it just once or a few times, so we want to get the result with minimal coding which may be dirty, non-performant and not suitable for landing in the Chromium code base, and we don’t want to write a permanent benchmark or add a permanent UMA.

Examples:

- To measure how often a situation happens in the real world, for trade-off decisions. Based on the result we can optimize the most frequently happening situations while accepting certain possible regressions in rarer cases
    - How often is a LayoutObject’s background obscured? Based on the answer we can decide whether the background obscuration optimization is worth the complexity.
- To measure effectiveness of algorithms and data structures
    - How many render surfaces are created with a new render surface creation algorithm?
    - How many LayoutBoxes have rare_data_? What are the reasons that we create them?

How to use

Prepare with trace events

We can skip this step if we already have trace events for the desired metrics in the supported format. Otherwise we add trace events. Trace events of event types ‘B’, ‘X’, ‘I” and ‘C’ are

supported. The following TRACE_EVENT macro formats are supported:

```
TRACE_COUNTER1("cat", "name", count)
TRACE_COUNTER2("cat", "name", "count_name1", count1, "count_name2",
count2)
TRACE_EVENT1("cat", "name", "count_name", count)
TRACE_EVENT2("cat", "name", "count_name1", count1, "count_name2",
count2)
TRACE_EVENT_INSTANT1("cat", "name", TRACE_EVENT_SCOPE_GLOBAL,
"count_name", count)
TRACE_EVENT_INSTANT2("cat", "name", TRACE_EVENT_SCOPE_GLOBAL,
"count_name1", count1, "count_name2", count2)
```

All integer values are interpreted as counts. Non-integer values are ignored. The metric name for each count is like "cat/name/count_name". Each value is reported as a scalar value with unit 'count'. If multiple values are reported for one metric name during a perf test cycle (typically a page load), the perf test result reporter will aggregate them, depending on "Field Value Column Name" in the ct job.

Counting occurrences

To count the number of occurrences of something interesting, instead of recording the count at a certain time (which may be difficult to find), you can simply add code like the following:

```
  if (something_interesting_occured)
    TRACE_COUNTER1("cat", "something", 1);
```

When running ct jobs, set "Field Value Column Name" to "count" or "sum" instead of "avg", and use "count" or "sum" instead of "avg" on results.html. If you use "count", the result will be the total number of this trace event, regardless of the value (the last parameter).

Run locally with top_25 pageset

```
tools/perf/run_benchmark generic_trace.top25 --trace-categories=cat
--trace-names=name
```

You can modify generic_trace.py locally to use other page sets.

Run on Cluster Telemetry (Google only)

1. If we added new trace events, upload the local change to gerrit (CL1), or land it if the trace events are also useful in the future.
2. Go to http://ct.skia.org.
3. Choose the Performance tab for comparison with and without a CL (CL2), or the Analysis tab for collecting data without comparison.

Note that because of a limitation of Cluster Telemetry frontend, CL2 needs to directly include (not just depend on) CL1.

4. Benchmark Name: generic_trace_ct
5. Benchmark Arguments: --trace-categories=cat --trace-names=name
   Append these after the default benchmark arguments.
6. Specify CLs
   a. Performance:
      Chromium Git patch: CL2
      Chromium Git metrics patch: CL1
   b. Analyze:
      Chromium Git patch: CL1
7. Field Value Column Name: This affects how multiple values of a metric are aggregated in the report.
   a. avg: the default, to get the average of the values
   b. sum: to get the sum of the values
   c. count: to get the count of the values. See also Counting occurrences
8. Fill the other fields as needed
9. QUEUE TASK

When the task finishes

a. Performance: you can browse the result directly in the result page
b. Analyze: the result is in csv format. You can save it and import it into a spreadsheet to analyze it.

Report format

Report format is controlled by perf test framework and cluster telemetry.

Sample reports (with this patch which adds some ad hoc metrics for counting layout objects with different features):

1. This HTML report was generated with the following local command:
   `tools/perf/run_benchmark generic_trace.top25 --trace-categories=blink --trace-names=counters`
   Check the Stories checkbox to show details for each page.

2. This CSV report was generated on Cluster Telemetry with an Analysis task.

3. TODO: sample report when we can apply an ad hoc metrics patch in a Cluster Telemetry Performance task.

Implementation

tools/perf/contrib/cluster_telemetry/generic_trace.py

blink_perf.py seems the best sample to implement a generic benchmark collecting trace event values. I'm not sure if I should use TBMv2, but the mechanism used by blink_perf.py seems straight-forward.

~~It's available in~~ This patch has landed in ToT.

ObjectCounter

This C++ class is for cases that no existing trace event exists for the desired measurement, and the number of objects is not easily counted otherwise (e.g. number of LayoutObjects).

It counts number of individual objects having each particular feature. It differs from UseCounter in that it counts individual objects instead of just appearance of features. It's only for ad hoc measurements, not suitable for production code because of its poor performance.

It's available in this patch. A sample usage is in this patch.

Future work

- Support timing metrics in addition to counts
- Custom page stories