Team and repository stats

Health metric monitoring: Google SpreadSheet
Public charts: Charts

"Management" tasks

- **SETTING UP REGULAR COMMUNITY CALLS**
  Notes:
  - Would guide the direction of development
  - It's a nice way to make the community "feel closer" to the projects

  Progress:
  Currently scheduled to Wednesdays 9AM UTC

- **Complete CII badge, on wiki pages**
  Caliper: https://bestpractices.coreinfrastructure.org/en/projects/2381
  Progress: Panyu 97%
  Problem Found: Missing versioning mechanism

- **Discuss requirements for PRs to be accepted**
  Notes:
  - Number of reviewer ACKs (with 2 maintainers confirmation)
  - Based on the impact of the PR, this should be dynamically decided, which means that statically 2 reviews should be enough, but for a major change all of the reviewers should approve first
  - Meet CI requirements, like successful build (naturally) and tests (different kind of coverage metrics) is lowest satisfaction, but reviewers should also inspect the correctness/meaningfulness of tests.

- **Involve more developers from non-Fabric platforms**
  Notes:
  - Even under Hyperledger, there are numerous DLT platforms that should be supported by Caliper.
  - It would make reviews easier if (core) developers of other platforms would be involved
  - Progress: Continuously invite maintainers or field experts of other projects. Victor will make a list of known experts for HLP project, so if we met problems on integrating with other project, we can know who we can reach out for help.

  Progress: Victor/ Nick will add more names this week. Table at the of the document

Short-term development tasks

Core Caliper-related

- ~~Add FAQ page to the document website~~
  Continue to extend the FAQ based on community questions

- **Add instructions of versioning and releasing rules and procedures**
  Priority: High
  Progress: Not started.
  Nick suggested to think about what is critical to a new version. Victor will think about that and make a plan for discussion in next meetings.

- Caliper is not running(under what configuration) correctly(what goes wrong) with high TPS(how much)
  Panyu please describe the problem by submitting an issue.
  Progress: issue has been submitted, pr is on the way, 70%

- **Add Iroha support**
  Priority: High
  Progress: Qinghui

- **Add Corda support**
  Priority: Middle
  Progress: Feihu Jiang

- **Define a simulator for high TPS Testing**
  Priority: High
  Progress: Not started.

- **Define a common signature for the adapters for invoking smart contracts**
  Notes:
  - Every network-related artifact should be managed by the adapter.
  - The client callback module should reference these by names/IDs. (The WIP Fabric adapter will follow this convention.)
  - The Fabric Node SDK pattern seems usable: a setting parameter for the invocation (this will be platform specific) and a timeout parameter (probably applicable for every platform)
  - This way the callback module code could calculate the current parameters of the smart contract (invoking identity, targets, arguments, etc.), and assemble these into the platform-specific invoke argument. So most of the module code would be platform independent.

  Priority: Medium

Progress: Needs look into different platforms, Attila already has a plan for Fabric. Attila can raise an issue talking about the plan, Feihu can look from a STL pov, and invite Iroha people for suggestion.

Not started yet(11/27)

- **Repo/module restructuring, aka, paving the road for the npm package**
  Notes:
  - Adopt similar repo structure like the Node SDK, and other npm best practices
  - There're starting to be more and more examples, maybe a caliper-samples repo would be useful later to separate the benchmarks from the code-base

  Priority: Medium
  Progress: Nick will ask advice from nodesdk and composer team since they 're more experienced in npm packaging. 30%
  Not changes yet. But we can do later, it will not strongly affect the npm scripts.

- **Align with metric definitions in PSWG whitepaper**
  Notes:
  - Caliper is made before the whitepaper come out
  - Check the current implementation to align with them
  - By Attila: only one latency tracking method is not implemented.

  Priority: Medium
  Progress: NA

- **Refactor Promise chains to async/await wherever possible**
  Notes:
  - Makes the code more readable
  - Results in clean error messages in case of unhandled exceptions
  - Easier "task-level" error handling instead of "chain-level" catches
  - Create issue first to collect feedback from community.

  Priority: High
  Progress: Attila (Github Issue), 80%(11/21),
  some modules (reporting, monitoring) are still not done

- **Introduction of the workspace concept**
  Notes:
  - Currently, most of the relative paths are resolved based on the caliper root dir. Golang chaincode paths are resolved according to GOPATH, which can be overwritten to the caliper root dir. But once caliper is installed as an npm package, this will be confusing.
  - For distributed zookeeper clients a self-contained "package" is needed that contains every artifact needed to run a test round.
  - Since the workspace is set explicitly, there's no guessing/assuming on the caliper side, the user will supply every path according to the workspace (or absolute paths).

  Priority: Medium
  Progress: NA

- Extract network queries to share among the Fabric adapters (channel joined or not)

Fabric-related

- **Refactor the adapter to use the Common Connection Profile**
  Notes:
  - Simplifies the code-base, as the SDK handles the assembling of object graphs and the management of client contexts.
  - Provides easy specification of (multiple) target nodes using only their names, which can be easily set even by the callback modules.
  - Fixes current shortcoming, like single channel test in a round, single endorsement policy for every chaincode, fixed target peer selection for a round, single orderer node support, etc.
  - Hides a lot of complexities behind a simple API, facilitating the support of new Fabric releases with minimal delay.
  - Merged as a new adapter (fabric11)

  Priority: High
  Progress: Attila Klenik; ~~code-base: ~70%->80%=>90%(10/24) => 95% (11.14)=> submit PR(11.21); tests: 0%; documentation: 10%~~ PRs pending

Medium-term development tasks

Core Caliper-related

- **Sketch some design documents about the internal communications/structure**
  Notes: These documents would help planning the modularization of the remaining components
  Priority: High
  Progress:

- **Modularize the monitoring module**
  Notes:
    - DLT deployments will probably be heterogeneous networks due to different participating organizations. Providing monitoring capabilities for a wide range of technology stack is cumbersome (and probably not the main purpose of Caliper).
    - The monitoring component should be pluggable, and wherever possible Caliper should utilize license-compatible third-party monitoring tools.

  Priority: Medium
  Progress: NA

- **Modularize the reporting module**
  Notes:
    - Currently, Caliper only supports the generation of HTML reports based on predefined metrics. This should be an expected functionality that reports on the different (PSWG) metrics, but Caliper should be also capable of providing detailed analysis.
    - Users might be interested in resource consumption during the tests, and not just the min/max values at the end.
    - Accordingly, the reporting module should be pluggable, and Caliper should provide at least a CSV export functionality about the gathered transaction data.
    - This would open up a way for the community to integrate numerous reporting and data analysis methods, especially in the age of big data.

  Priority: Low
  Progress: NA

Fabric-related

- **Fork a separate process(es) from the Fabric adapter, that only handles event processing, like in PR 124**
  Notes:
    - Timing critical measurements are not possible under high load due to the single threaded nature of Node.js. Event processing in a separate process

would take off a great load from the “transaction submitting” process, moreover, also decreasing the interference with rate controller precision.

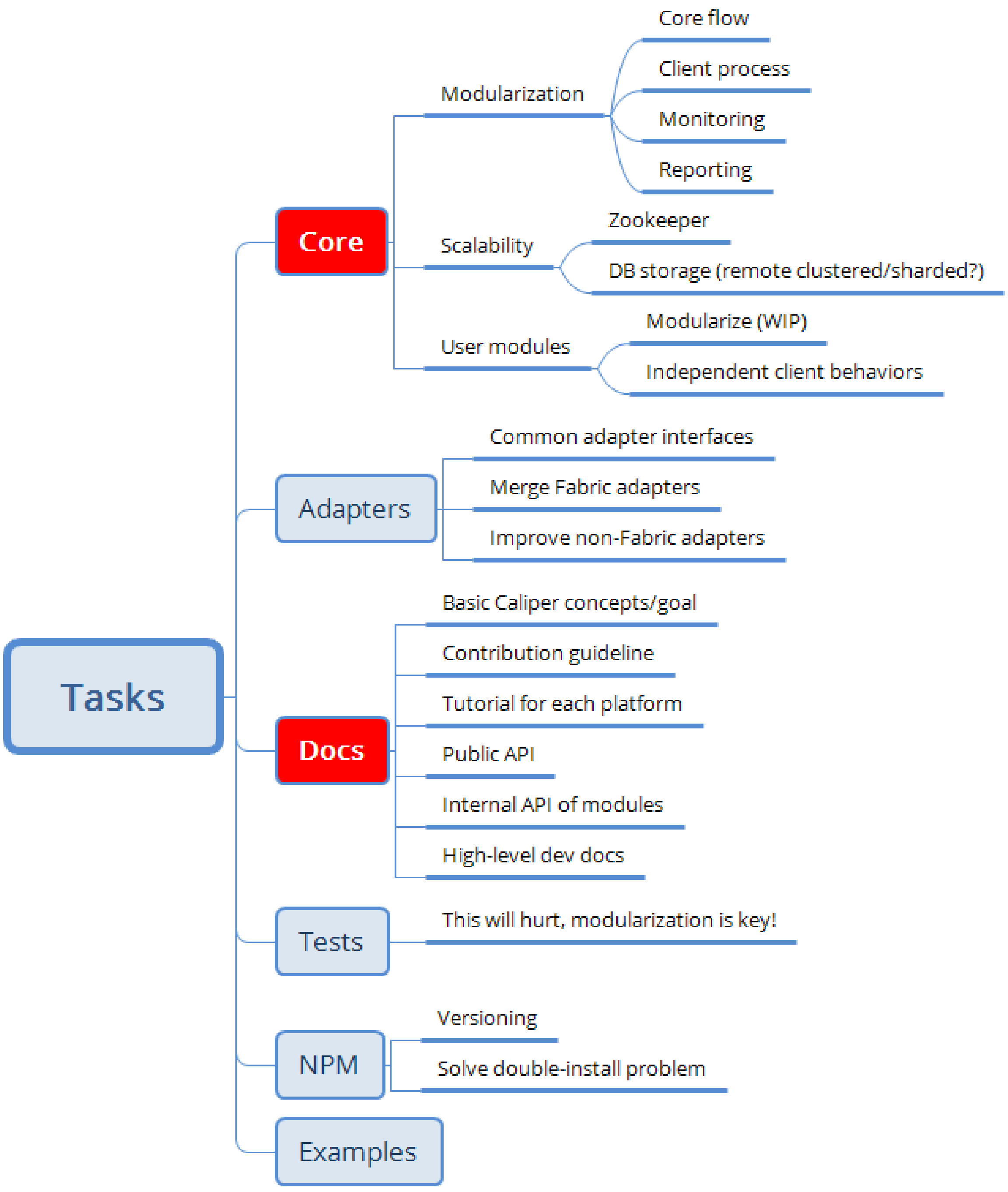
- However, both end of the scale must be supported: efficient/fast event processing by connecting to only one needed channel event hub; connecting to every channel event hub to calculate metrics like transaction latency with a 100% network threshold, as discussed by the PSWG.
- Maybe a general architectural pattern can be derived and applied for the other supported platforms as well.
- PR124 is merged into the current Fabric adapter

Priority: Medium/High
Progress: NA

Long-term development tasks

Core Caliper Related

- Extend Caliper‘s scope
    - Add system test for consensus effectiveness
- Other DLT support
    - which?


Tasks
Core
Modularization
Core flow
Client process
Monitoring
Reporting
Scalability
Zookeeper
DB storage (remote clustered/sharded?)
User modules
Modularize (WIP)
Independent client behaviors
Adapters
Common adapter interfaces
Merge Fabric adapters
Improve non-Fabric adapters
Docs
Basic Caliper concepts/goal
Contribution guideline
Tutorial for each platform
Public API
Internal API of modules
High-level dev docs
Tests
This will hurt, modularization is key!
NPM
Versioning
Solve double-install problem
Examples

Platform supports

Here we collect the remarks about the different platform supports, for example, the missing but desired features.

Fabric

Documentation: https://hyperledger.github.io/caliper/docs/Fabric_Configuration.html
TBD

Fabric CCP

PR in progress

Sawtooth

Documentation: https://hyperledger.github.io/caliper/docs/Sawtooth_Configuration.html
TBD

Iroha

Documentation: https://hyperledger.github.io/caliper/docs/Iroha_Configuration.html
TBD

Burrow

Documentation: https://hyperledger.github.io/caliper/docs/Burrow_Configuration.html
TBD

Contacts table

Project	Expert	Contact
Fabric	Gari Singh	
	Yacov Manevich	
	Jason Yellick	
	Dave Kelsey	@davidkel
Sawtooth	Peter Schwartz	
	Dan Middleton	
Iroha	Andrei Lebedev	
	Sara G	
Composer	Nick Lincoln	@nkl199
	Simon Stone	@sstone1
Burrow	Silas Davis	
Indy	Alexander Shcherbakov	
	Sergey Khoroshavin	@sergey.khoroshavin
Ethereum	Nick Johnson	
IOTA	Dominik Schiener	
Cello	Baohua Yang	
Explorer	Nik Frunza	
Quilt	David Fuelling	

Troubleshooting / FAQ

Note: A FAQ page already exists. Here we should collect additional entries that should be added to it (before submitting it as a PR, since that's a slower process).

Documentation TODOs

Hyperledger Bootcamp Task

Improve the logging support

Difficulty: Easy

Context

Caliper utilizes the winston logging package to provide a flexible and common mechanism for modules for logging. Winston provides some core transports/target for logging. Currently, Caliper uses the following syntax in its default configuration file to configure targets:

```yaml
core:
  log-file:
    debug: log/caliper.log
    info: console
```

This will result in two targets: a file logger with debug level filtering and a console logger with info level filtering.

Goal

A more flexible approach would be to allow the definition of an arbitrary number of loggers (denoted by a unique name/attribute in the configuration), with configurable targets and target/winston-specific options (corresponding to the target API):

```yaml
core:
  logging:
    consolelogger:
      # indicates the Console transport
      target: console
      # console transport specific options, mostly optional
      level: info
      colorize: true
    filelogger:
      # indicates the File transport
      target: file
      # file transport specific options, mostly optional
      level: debug
      filename: path/to/logfile.log
```

This is a flexible way to define as many logging targets as we want with arbitrary settings, plus adding support for new types of logging targets (http, MongoDB, Syslog, etc) is easier, just check the target property, and add a new transport creation code for the new target.

Corresponding documentation should also be provided, detailing the available target types and the general logging mechanism (how to configure loggers, how to create loggers in a module).

Guide

The logging/winston-related codes are in the `src/comm/util.js` file. Specifically, the `Util.getLogger` function should be improved according to the above specification and examples. Moreover, it would be nice to extract the logging-related functions into their own utility file (`src/comm/logging-util.js`), similarly to `src/comm/config-util.js`

Improve the configuration support

Difficulty: Easy

Context

Caliper uses a hierarchical configuration mechanism provided by the nconf package. The priority between the different configuration sources are the following:

Runtime > Environment variable > Command line > Configuration file

The provided environment variables (conventionally written in uppercase with underscore separators) are made lowercase and the underscores are changed to dashes. For example, `CORE_SOMESETTING=10` can be queried through the `"core-somesetting"` configuration key.

The configuration file allows nested objects, that are flattened to separate keys for each "leaf" attribute. For example, the value of `somesetting` in

```
core:
 someSetting: 10
```

can be queried through the `"core:someSetting"` configuration key. These two examples demonstrate the inconsistency of the configuration mechanism: someSetting cannot be overridden by environment variables since `CORE_SOMESETTING` will be mapped to `"core:somesetting"` (note the missing camel casing).

Goal

The goal would be to use a common format for settings from every source. The proposed format is the following: *all lowercase letters* for the setting hierarchies, *separated by dashes*. So the previous examples both could be queried with the same configuration key, `"core-somesetting"`, and could be overridden by the environment variable `CORE_SOMESETTING=10`, or through the command line with `--core-somesetting=10`.

Moreover, to prevent name collisions with settings outside of Caliper (especially for environment variables), the `"caliper-"` prefix should be used for every settings key.

It is also important to parse well-know string values (e.g., 'false', 'true', '3', '5.1' or JSON values) as their corresponding type (e.g., boolean, number, or object), both for command line arguments and environment variables.

Corresponding documentation should also be provided that describes the configuration mechanism of Caliper.

Guide

Using the `"caliper-"` prefix requires the modification of the default configuration file. Everything should be enclosed in a top-level `"caliper"` attribute. This way nconf automatically will append the prefix, since it's part of the configuration hierarchy. For example:

```
caliper:
  core:
    somesetting: 10
```

The environment variables are set by the users, so it's their responsibility to correctly override, for example, the `"caliper-core-somesetting"` value by setting the `CALIPER_CORE_SOMESETTING` environment variable.

Consistent naming can be achieved by using all lowercase names in the configuration file, plus setting the appropriate nconf option (`options.logicalSeparator`), so it uses dashes as a separator when parsing a configuration file.

Parsing well-known string values can be achieved by using the appropriate settings for nconf.

Modularize the rate controllers

Difficulty: Easy

Context

Caliper uses rate controllers to signal the user callback modules when they can submit transactions. Currently, rate controllers have to provide/export a class containing some functions to be usable by Caliper (like a constructor, an init, apply and end functions).

Also, Caliper can only use rate controllers that are explicitly listed in the rate controller gateway. This limits the easy extensibility of the mechanism.

Goal

The proposal is to modularize the rate controller implementations in the following way: a module implementing a rate controller should only expose/export a factory function, like `createRateContoller(bc, opts):object`, that should return an object that has the aforementioned functions. This way the rate controller developer would have explicit control over the lifetime of a rate controller object.

Moreover, external rate controllers also should be supported, i.e., the `src/comm/rate-control/rateControl.js` class should be able to resolve paths to external JS files, provided that they export the necessary factory function.

Corresponding documentation should also be provided that describes the general API of rate controllers.

Guide

The existing rate controller files (in the `src/comm/rate-control` directory) should be modified to export the aforementioned factory function instead of the rate controller class. Additionally, the `src/comm/rate-control/rateControl.js` constructor should be modified to create the rate controllers through the factory function.
Also, if it encounters an unknown rate controller type that is a path to a JS file, it should try to access its factory function if it's provided.

Modularize the user test modules

Difficulty: Easy

Context

Caliper uses user modules (or user test modules) to allow arbitrary workload logic to be plugged in. The user module must export the three required functions to be able to interact with Caliper. This design is not object-oriented and should be refactored.

Goal

The proposal is to require only an exported factory function (e.g., `createUserModule`) from the user module JS file. This way the developer has more freedom to structure the implementation and explicitly manage the life-cycle of the user module instance.

Corresponding documentation must also be provided that describes the general API of user modules.

Guide

The `src/comm/client/local-client.js/run*` functions must be refactored to use the new factory function of the referenced modules.
Additionally, the provided example user modules (in the `benchmark/` directory) must also be refactored.

Improve the adapter life-cycle management

Difficulty: Intermediate

Context

Caliper is comprised of some basic building blocks: platform adapters, rate controllers, user modules, client processes, and the main process.
While rate controllers and user modules are only instantiated by client processes, platform adapters can exist in two different environments: both in the main process and in the client processes.

The current life-cycle of the platform adapters (PA) in the main process (MP) is the following:

1. MP creates PA
2. MP initializes PA

3. MP instructs PA to install the (optionally) configured smart contracts
4. MP spawns the client processes

The current life-cycle of the platform adapters (PA) in the client processes (CP) is the following:

1. CP creates PA
2. Before a test round, CP requests the current context from PA
3. CP performs a test round
4. After a test round, CP releases the current context through PA
5. If there are more test rounds, repeat from 2.

This life-cycle management/protocol is not granular enough from a resource management point-of-view.

Goal

The proposal is to extend the introduced life-cycle of platform adapters to include additional communication between the client/master process and the adapter. For example:

The proposed life-cycle of the platform adapters (PA) in the main process (MP):

1. MP creates PA
2. MP initializes PA
3. MP instructs PA to install the (optionally) configured smart contracts
4. MP spawns the client processes
5. **MP disposes of PA after the clients finished their work**

The proposed life-cycle of the platform adapters (PA) in the client processes (CP):

1. CP creates PA
2. **CP initializes PA “in client mode”**
3. Before a test round, CP requests the current context from PA
4. CP performs a test round
5. After a test round, CP releases the current context through PA
6. If there are more test rounds, repeat from 2.
7. **CP disposes of PA “in client mode”**

Guide

The adapter interfaces need to be extended to support the new life-cycle phases. Specifically, the following classes need to be modified:

- src/comm/blockchain.js
- src/comm/blockchain-interface.js
- Platform adapters in src/adapters
- src/comm/bench-flow.js
- src/comm/client/local-client.js