iExec platform documentation: Trust in the PoCo

Hadrien Croubois
hc@iex.ec

Part 1

Overview of the consensus

According to the PoCo, consensus is performed for each task t (described by a `bytes32 taskid`). Consensus on a task involves one or multiple workers (w) contributing to the task. Contributions $H_t(w)$ are specific to a worker/task. While the contributions are protected by the PoCo, in the consensus buidling phase we consider the result's hash which is common to workers who computed the same value.

Workers have a score $s(w)$, stored on-chain, which corresponds to the number of passed checkpoint (see later). This score will determine the weight of the workers contribution an thus the number of contributions needed to achieve consensus.

1.1 Notations

Task:	t	
Worker:	w	
Worker score:	$s(w)$	
Worker likelihood of good contribution:	$P^{good}(w)$	
Worker likelihood of bad contribution:	$P^{bad}(w)$	$= 1 - P^{good}(w)$
Worker weight:	$\tilde{P}(w)$	$= \frac{P^{good}(w)}{P^{bad}(w)} = \frac{1}{P^{bad}(w)} - 1$
Worker contribution:	$H_t(w)$	
Potential results:	R_t	$= \{H_t(w) \mid w \text{ contributed to } t\}$
Result correctness likelihood:	$P_t^{good}(r)$	$= \prod_{H_t(w)=g} P^{good}(w)$
Result fault likelihood:	$P_t^{bad}(r)$	$= \prod_{H_t(w)=g} P^{bad}(w)$
Result weight:	$\tilde{P}_t(r)$	$= \frac{P_t^{good}(r)}{P_t^{bad}(r)} = \prod_{H_t(w)=g} \tilde{P}(w)$
Result credibility:	$Cr_t(r)$	

Table 1: Notations used in the PoCo's consensus

1.2 Workers weight caracterisation

Following Sarmenta's proposition, $P^{good}(w) = 1 - \frac{f}{s(w)}$, and thus $P^{bad}(w) = \frac{f}{s(w)}$.

$$\tilde{P}(w) = \frac{s(w)}{f} - 1 \tag{1}$$

With $f = 0.2$ (proposed by Sarmenta), we have $\tilde{P}(w) = 5 \times s(w) - 1$.

1.3

Consensus caracterisation

Among the potiential results, a result is accepted if $Cr_t(r) > 1 - \frac{1}{trust}$. New worker are required to contribute until this threshold is reached. Sarmenta's definition of the result credibility can be rewritten to be a function of the results' weights:

$$\begin{aligned} Cr_t(r) &= \frac{P_t^{good}(r) \prod\limits_{h \in R_t \setminus g} P_t^{bad}(h)}{\sum\limits_{h \in R_t} \left(P_t^{good}(h) \prod\limits_{k \in R_t \setminus h} P_t^{bad}(k) \right) + \prod\limits_{h \in R_t} P_t^{bad}(h)} \\ &= \frac{\frac{P_t^{good}(r)}{P_t^{bad}(r)} \prod\limits_{h \in R_t} P_t^{bad}(h)}{\sum\limits_{h \in R_t} \left(\frac{P_t^{good}(h)}{P_t^{bad}(h)} \prod\limits_{k \in R_t} P_t^{bad}(k) \right) + \prod\limits_{h \in R_t} P_t^{bad}(h)} \\ &= \frac{\tilde{P}_t(r) \prod\limits_{h \in R_t} P^{bad}(h)}{\left(1 + \sum\limits_{h \in R_t} \tilde{P}_t(h) \right) \prod\limits_{h \in R_t} P^{bad}(h)} \end{aligned}$$

$$Cr_t(r) = \frac{\tilde{P}_t(r)}{1 + \sum\limits_{h \in R_t} \tilde{P}_t(h)} \tag{2}$$

Part 2

Smart contract implementation of the consensus

With each contribution, we verify if the consensus is achieved. This requires the `IexecHub` to keep track of the consensus weights. For each task we keep track of each potential result's weight and of the total weight:

$$\begin{aligned} \tilde{P}_t(r) &\quad \Leftrightarrow \quad \texttt{m\_groupweight(bytes32 => bytes32 => uint256)} \\ 1 + \sum_{h \in R_t} \tilde{P}_t(h) &\quad \Leftrightarrow \quad \texttt{m\_totalweight(bytes32 => uint256)} \end{aligned}$$

As a contribution only increasesses the credibility of the results it's weighting for, we only have one potential result to check. Consensus is achieved if and only if:

$$\tilde{P}_t(g) \times trust \geq \left(1 + \sum_{h \in R_t} \tilde{P}_t(h) \right) \times (trust - 1) \tag{3}$$

If this condition is reached, the task moves to the reveal phase, which means no new contributions are accepted and workers who contributed toward the winning result are required to reveal.

Part 3

Worker's score dynamics

A worker's score is tight to its address. The score is recorded on the blockchain, and evolves based on the worker's contributions.

Sarmenta's describes the score $s(w)$ as the number of checkpoints succesfully passed. In our context, this corresponds to the number of successive successfull contributions that were part of a multiparty consensus. This means that if the consensus is achieved with a single contribution, the worker who provided this contribution does not get any score reward.

If a workers provides a bad contribution, either a bad value or a value that was not correctly revealed during the reveal phase, the workers loses some reputation. Sarmenta's approach is to reset the score to 0.

3.1

Designing stable cycles

A stable score cycle is a balance between the impact of the score and the subsequent loss of score in case a bad result is submitted. If a given score leads to a high result credibility, then any error must result in a significant loss of score. On the other hand, if the score the same amount of score is viewed as a lower credibility, than we do not have to remove as much. Choosing a score dynamic is a balance between the hability for worker to gain weight quickly and their the amount they lose in case of bad behaviour. This can be modified using the f parameter.

Our objective is to build a score policy that will produce stable cycles so the score of a worker corresponds to it's reliability.

Lets consider a stable cycle where a worker provides a bad contribution he loses a fraction $\frac{1}{k}$ of it's score. We have a cycle that is stable if $MTBF = \frac{s(w)}{k}$, which implies:

$$\tilde{P}(w) = \frac{s(w)}{k} - 1 \tag{4}$$

Exceptions have to be made for low score as $\tilde{P}$ has to be scrictly bigger than 1 for the result credibility to grow, but this policy fits the requirements expressed by the requester (*trust*). We notice that this formula for worker weight is similar to Sarmenta's, except with values of f larger then 1.

$$s(w) \mapsto \begin{cases} s(w), & \text{for unreplicated correct contributions} \\ s(w)+1, & \text{for replicated correct contributions} \\ s(w) * \frac{k-1}{k}, & \text{for incorrect contributions} \end{cases} \tag{5}$$

The choice of k is a choice of balance between the reactivity and the stability of the model:

- A high reactivity (for small k) means worker gain credibility very fast as their score goes up. On the other hand, bad behaviour causes large score loss, creating unstability. This favors new workers that can quickly gain weight. Older worker risk a lot in case of bad contribution, but will recover from it pretty quickly. This is what Sarmenta proposes by using $k \equiv f = 0.2$. Note that $k < 1$ causes the worker to lose all of its score for any error.
- A low reactivity (large value of k) gives the worker a more stable score. On the other hand, gaining this score on the first place would be more difficult. This favors old workers who risk less in case of accidental error or attack but increasses the difficulty of new workers to gain weight in the paltform.

Sarmenta's approach ($f = 0.2$) is visible in Fig. 1. It correspond to a high platform reactivity. With the credibility of a worker growing very fast we see that it is often overestimated. The worker with reliability 99.99% ($1e^{-4}$ error rate) almost reaches an estimated error rate of $1e^{-}5$ before an error is reached and its score is reset.

iExec uses $k = 3$, which is a good middleground. The corresponding behaviour is visible in Fig. 2. In this approach, workers credibility grows slower than in Sarmenta's approach but there are also stronger incentives to stay in the platform. Whe workers reliability is also more accuratelly predicted. Fig. 2 shows that the worker's score isn't commonly overestimated and isn't completelly lost when an error is detected.

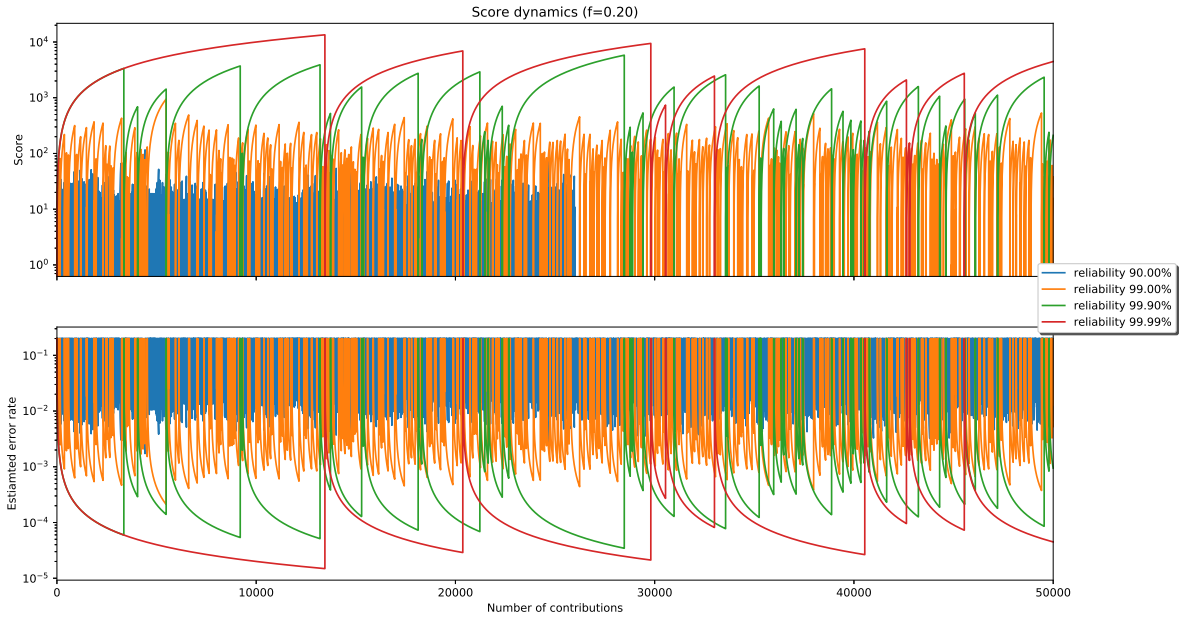

Figure 1: Evolution of the score and predicted reliability of workers of various reliability over 50000 tasks (Sarmenta's checkpoint version).

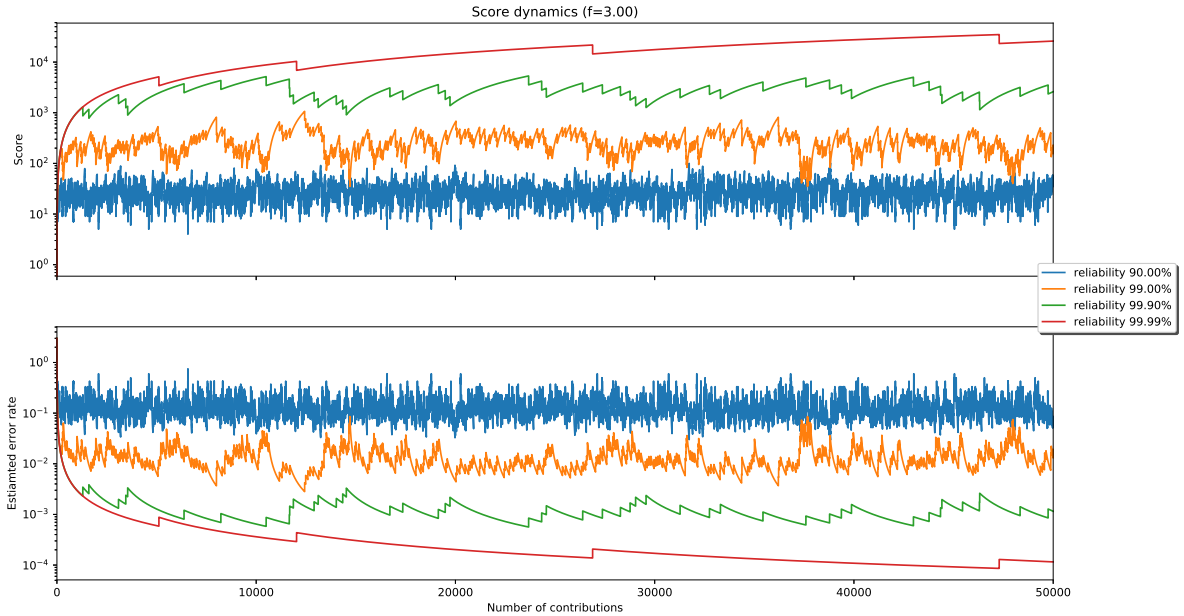

Figure 2: Evolution of the score and predicted reliability of workers of various reliability over 50000 tasks (Proposed version, $k = 3$).

Part 4

Requesting a trust level

Thanks to the accurate evaluation of the workers reliability, we can evaluate the likelihood of a proposed result being correct and use it, as described earlier, to accept results depending on the user required quality of service.

When submitting a requestorder, which describes a desired execution, the user specifies a *trust* value. This value determines the confidence level the the user wants the PoCo to enforce. This *trust* value can easily be translated into a percentage.

Trust (t)	PoCo enforced confidence level ($1 - \frac{1}{t}$)
0	0%
1	0%
2	50%
100	99%
10000	99.99%
1000000	99.9999%

Table 2: Translation of *trust* value in confidence levels

Using a low trust level can make sense:

- If the user trusts a workerpool and doesn't want the benefit of PoCo's trust layer;
- If the application is not deterministic (consensus cannot be achieved);
- If security is enforced through another mechanism (such as hardware enclave).

These elements are not mutually exclusive. When a user requires the execution of a non-deterministic application, the use of a low trust level (by passing the PoCo replication layer) will be combined with the use of harware enclaves to ensure a trustworthy result is produced.

On the order hand, when an application is deterministic and extra trust is required, for example when results are processed by the blockchain, the PoCo's trust layer can be used to guaranty a certain quality of service.

It is up to the requester to determine whether the want to use this mechanism or not.