MODiCuM Specification

Scott Eisele, Taha Eghtesad, Nicholas Troutman, Aron Laszka, Abhishek Dubey

1 Rules

1.1 Matching

We can match two offers if they satisfy the following conditions: Job offer limit variables must be lower than the resource offer cap variables. Job offer max variables must be higher than the resource offer price variables. Additionally, there should be a mediator with the same architecture of JO which is trusted by both JC and RP.

$$RO.instruction\_capacity \geq JO.instructions\_limit \tag{1}$$
$$RO.ram\_capacity \geq JO.ram\_limit \tag{2}$$
$$RO.local\_storage\_capacity \geq JO.local\_storage\_limit \tag{3}$$
$$RO.bandwidth\_capacity \geq JO.bandwidth\_limit \tag{4}$$
$$RO.instruction\_price \leq JO.instruction\_max\_price \tag{5}$$
$$RO.bandwidth\_price \leq JO.max\_bandwidth\_price \tag{6}$$
$$JO.architecture = RP.architecture \tag{7}$$
$$JO.directory \in RP.trusted\_directories \tag{8}$$

$$\begin{aligned} \exists M: \ & M \in (JC.trusted\_mediators \cap RP.trusted\_mediators) \\ & \wedge M.architecture = RP.architecture \qquad (9) \\ & \wedge JO.directory \in M.trusted\_directories \qquad (10) \end{aligned}$$

$$current\_time + RP.time\_per\_instruction \cdot JO.instruction\_limit \leq JO.completion\_deadline \tag{11}$$

1.2 Payment

When the JC or the RP wants to post an offer for a job or resource, it will pay a deposit value to prevent it from cheating on the platform. This deposit value is a function of the posted offer which is more than the price of the job plus mediation. As the price of job or mediation is unclear at the time of posting the offer, a static $penalty\_rate \gg 1$ is applied as security deposit.

After the job is finished and both the JC and RP agree on the outcome, JC will receive the deposit minus the cost of the job and the RP will receive the deposited value plus the cost of the job.

In case of disagreement, the match will go for mediation. The mediators have a fixed price for their resources. After the submission of their verdict, the deposit of the party who is at fault will be forfeited in favor of the winner and winner will receive both the deposits minus the cost of mediation.

$$\begin{aligned} JO.deposit &= (instruction\_limit \cdot instruction\_max\_price + \\ &\quad bandwidth\_limit \cdot bandwidth\_max\_price) \cdot penalty\_rate \end{aligned}$$

$$\begin{aligned} RP.deposit &= (instruction\_cap \cdot instruction\_price + \\ &\quad bandwidth\_cap \cdot bandwidth\_price\ ) \cdot penalty\_rate \end{aligned}$$

$$\begin{aligned} job\_cost &= result.instruction\_count \cdot resource\_offer.instruction\_price + \\ &\quad result.bandwidth\_usage \cdot resource\_offer.bandwidth\_price \end{aligned}$$

2 Data Structures

The platform is a composition of a smart contract, resource providers, job creators, mediators, and directories.

```
Platform {
   mediators: Mediator[],
   resource_providers: ResourceProvider[],
   job_creators: JobCreator[],

   resource_offers: ResourceOffer[],
   job_offers: JobOffer[],
   matches: Match[],
   results: JobResults[],
   penaltyRate : uint
}
```

All information is stored within well-known storage services. The platform itself is agnostic to a particular service. However, it is assumed that all storage providers, which we call Directories, support SFTP/SCP Gateways for uploading and downloading documents. To use SFTP, the client must authenticate itself and then upload or download files using fully qualified name. A file once uploaded can be shared with other users if their user-id is known.

During the initial setup, we will support NFS based storage service. The resource providers, job creators and mediators will be authenticated using uuid. [please see NFS authentication for further notes.]

In the platform, we identify the directories by using their URL, which specifies the server address and the mount point. Anyone using the service has to ensure their own access to the directory.

The platform can support multiple architectures. However, initially it will only have amd64 and armv7.

```
enum Architecture {
    amd64,
    armv7
}
```

2.1 Entities

```
JobCreator {
    trusted_mediators: Mediator[]
}


ResourceProvider {
    trusted_mediators: Mediator[],
    trusted_directories: address[],
    arch: Architecture,
    time_per_instruction: uint
}
```

`trusted_directories` lists Directory entities which the ResourceProvider can use for both download and upload (the latter typically requires having some account on the Directory). `time_per_instruction` is an estimate of how much time the ResourceProvider needs on average to execute an instruction, which is used when matching to check if the ResourceProvider could finish the job before the deadline.

2.1.1 Mediators

Mediators are used for arbitration. The mediator structure looks very much like the resource provider, except that it includes additional costs for mediation.

```
Mediator {
    supported_arch: Architecture,
```

```
    instruction_price: uint,
    bandwidth_price: uint,
    docker_bandwidth_price: uint
}
```

2.2 Offers

```
JobOffer {

        jobHash: hash,

        job_creator: JobCreator,

        URI: string,
        arch: Architecture,

        instructions_limit: uint,
        ram_limit: uint,
        local_storage_limit: uint,
        bandwidth_limit: uint,

        instruction_max_price: uint,
        max_bandwidth_price: uint,

        completion_deadline: datetime

        deposited_value: uint,

    }
```

`URI` gives the location of the job-specific files on the Directory. The limits specify how many instructions the JobCreator is willing to pay for (after executing this many, the ResourceProvider can give up and still get paid), how much RAM, local storage, and bandwidth the ResourceProvider may have to use (again, after reaching these limits, the Resource Provider may stop). `instruction_max_price` specifies the maximum price per instruction that the JobCreator accepts, `max_bandwidth_price` specifies the maximum price per downloaded / uploaded byte (for the job, not for layer) that the JobCreator accepts. `max_docker_bandwidth_price` specifies the maximum total price that the JobCreator is willing to pay to the ResourceProvider for downloading images that the ResourceProvider does not have. `time_to_completion` is the deadline of RP for submitting the solution. If it happens that the RP has missed the deadline, the JC can call a `timeout` function which will punish the RP by taking all of it's deposit and giving it to the JC. Before an offer is matched, the participant can cancel the offer.

```
    ResourceOffer {

        instruction_price: uint,
        instruction_cap: uint,

        memory_cap: uint,
        local_storage_cap: uint,

        bandwidth_cap: uint,
        bandwidth_price: uint,

        deposit_value: uint
    }
```

Prices are per instruction or per byte. Capacities specify what resources the ResourceProvider has, and they are used as constraints for matching. All of the participants can deposit as much as they want as long as it is more than the required `JO/RP.deposit`. `deposit_value` is the variable that holds the amount of deposited value for each offer.

2.3 Results

```
JobResult {
    match: Match,
    status: ResultStatus,
    URI: string,
    instruction_count: uint,
    bandwidth_usage: uint,
    timestamp: datetime
}
```

`URI` gives the location of the result on the Directory. `instruction_count` is the number of instruction that were executed by the ResourceProvider. `bandwidth_usage` is the number of bytes downloaded / uploaded by the ResourceProvider for the job (not counting the download of Docker layers). The JC has a specific deadline for responding to a result. Whether to approve or decline it. If the deadline is missed the RP can accept the result instead of the JC.

```
MediatorResult {
    status: ResultStatus,
    uri: string,

    matchId: uint,

    hash: uint,

    instructionCount: uint,
    bandwidthUsage: uint,

    verdict: Verdict,
    faultyParty: Party
}
```

The platform on itself cannot determine which party is cheating merely based on the `hash` of the Mediator's Results. For example, the smart contract can not check whether the job was correctly posted to the directory by the JobCreator or the result was correctly posted to directory by the ResourceProvider. Therefore, it is the responsibility of the Mediator to decide who should be punished in the ecosystem. `verdict` is the reason for deciding on the cheating party, and `faultyParty` is the cheating party.

```
Match {
    resource_offer: ResourceOffer,
    job_offer: JobOffer,
    mediator: Mediator
}


enum ResultStatus {
    Completed,
    Declined,
    JobDescriptionError,
    JobNotFound,
    MemoryExceeded,
    StorageExceeded,
    InstructionsExceeded,
    BandwidthExceeded,
    ExceptionOccured,
    DirectoryUnavailable
}
```

`Completed` means that the ResourceProvider finished the job successfully and posted the results on the Directory. `JobDescriptionError` means that there is an error in the job description. `MemoryExceeded`, `InstructionsExceeded`, `BandwidthExceeded` mean that the job exceeded the limits specified in the JobOffer. `ExceptionOccured` means that an exception was encountered while executing the job, while `DirectoryUnavailable` means that the ResourceProvider is unable to post results because the Directory is unavailable.

```
enum Verdict {
    ResultNotFound,
    TooMuchCost,
    WrongResults,
    CorrectResults,
    InvalidResultStatus
}
```

This is the reason that the Mediator chose to punish an actor. `ResultNotOnDirectory` means that the ResourceProvider did not put the results in the directory. `TooMuchCost` means that the ResourceProvider charged too much for the job. `WrongResults` means that the ResourceProvider provided wrong results. `CorrectResults` means that the ResourceProvider completed the job correctly and the JobCreator should be punished by sending the result for mediation. `InvalidResultStatus` means that the ResourceProvider's mentioned ResultStatus is wrong. For example, it said that the job description was invalid and returned `JobDescriptionError`, but the description was correct.

```
enum Party {
    ResourceProvider,
    JobCreator
}
```

These are the trustless parties in the ecosystem. Mediators should specify who cheated in a job and should be punished by specifying one of these parties.

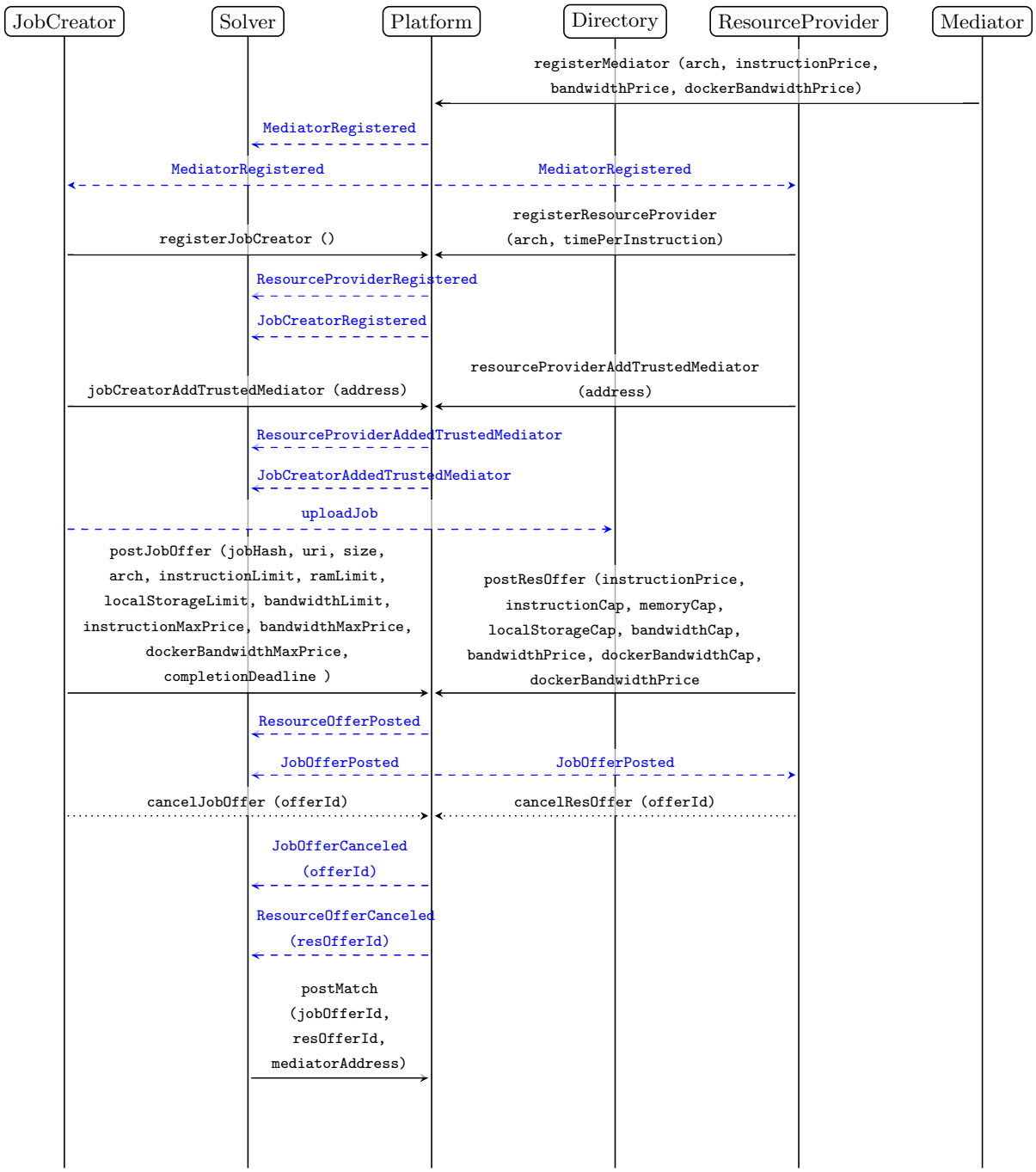

Figure 1: Sequence diagram. Part I. Sequence diagram showing the outsourcing of a single job. Black arrows are function calls to the smart contract. Blue dashed lines are events emitted by the smart contract. Black dotted lines are optional function calls. Red lines are optional calls that are required in case of disagreement between RP and JC.

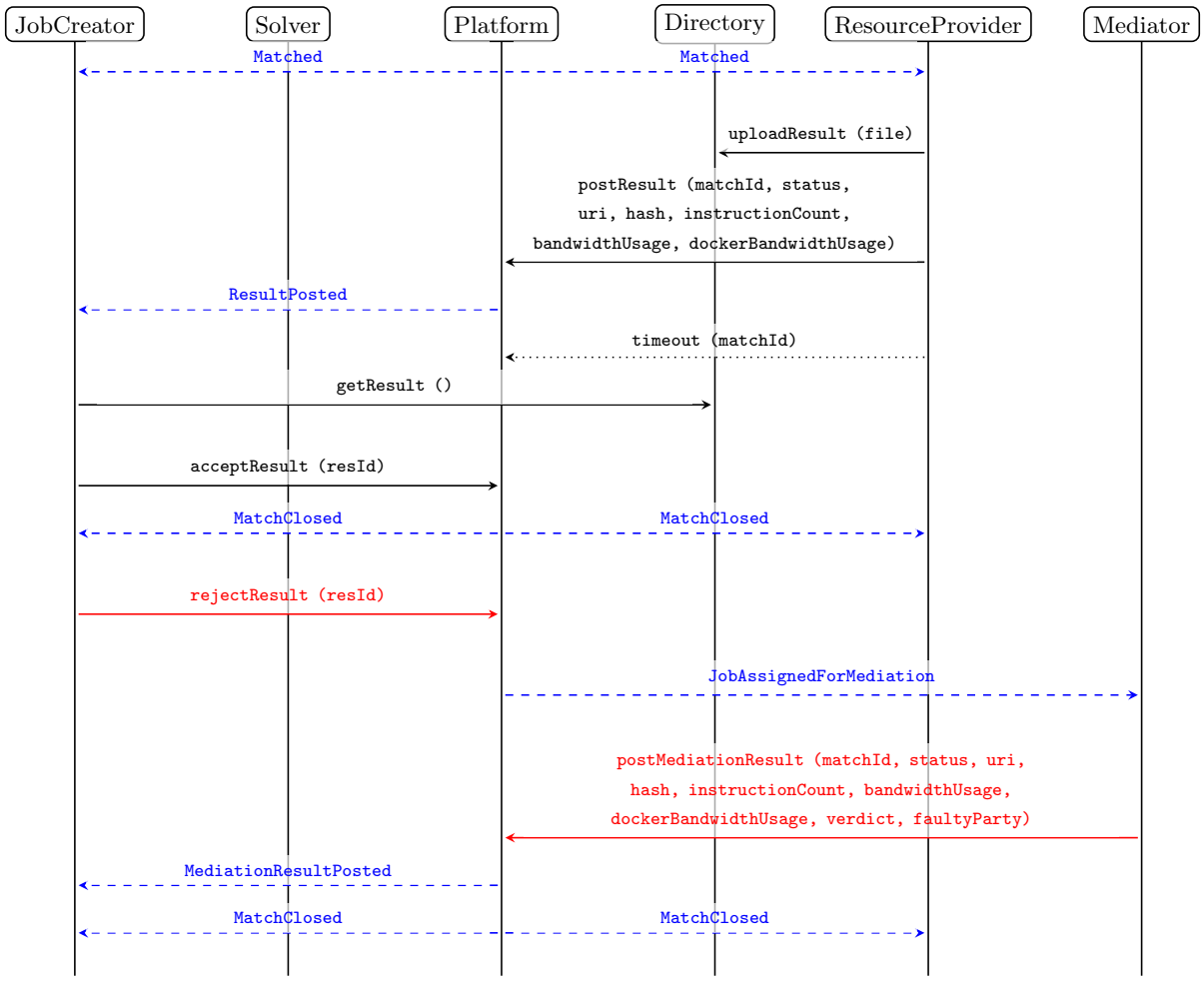

Figure 2: Sequence diagram. Part II