PROPOSAL

“Marx”

International Justice League of Super Acquaintances (IJLSA)

Brody Concannon
Nathan Karasch
Stefan Kraus
Gregory Steenhagen

Goal

We will create a distributed computing map engine that accepts a program written in Python 2.7, divides it into separately executable portions, distributes the portions to computers on the same network for execution, and then assembles and returns the results to the sender of the program.

Discussion

Project Structure

Project Marx will be comprised of three main parts:

1. **A Centralized Distribution Server**
This will be run as the main server to accept incoming *Tasks* from *Tasker Clients*, divide the code into chunks, and distribute the chunks to *Worker Clients* that are broadcasting a "Ready for Work" signal to the server. The server then accepts results, logs, and artifacts from the *Worker Clients* and sends them back to the initiator of the specific *Task.*

2. **Tasker Clients**
These clients will login to the server and send the server work *Tasks* to complete in the form of python files. They then receive the results back from the server upon completion of the work or upon completion of portions (chunks) of the work. They should satisfy the minimum computer requirements outlined in the "Required from Clients" section

3. **Worker Clients**
These clients execute any code chunks received from the server, returning the results upon completion. When they are running the application and have a status of "Ready for Work," they may receive program chunks from the *Centralized Distribution Server* to be run. They then execute the code, record any logs or artifacts from execution, and then send them back to the server. The *Worker Clients* should satisfy the minimum computer requirements outlined in the "Required from Clients" section.

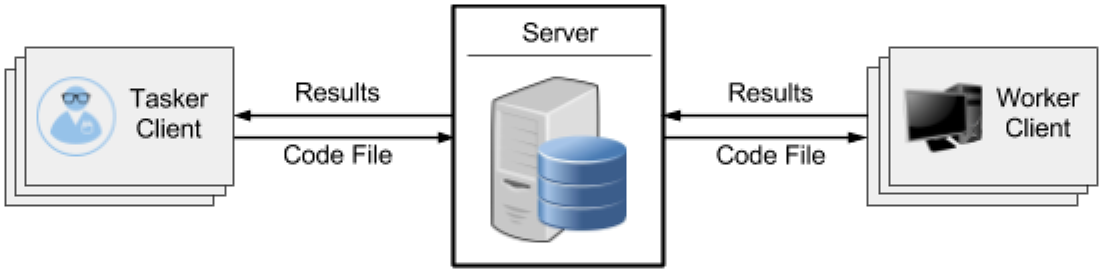


Figure 1: A simplified block diagram of the flow of information from Tasker Client to Worker Client and back.

Benefits to Customers

The current options for Universities that wish to do computationally complex research are mostly limited to either dedicated supercomputers or cloud computing services. Dedicated supercomputers are a large cost to setup and maintain, and can cause a central point of failure that can cost a great amount of lost work if the system goes down. Cloud computing services have great guarantees of uptime and redundancies with no investment in infrastructure, but come with significant monthly costs. Our project aims to have a minimal startup cost, and nearly no additional maintenance cost depending on the available infrastructure.

Deliverables

We will deliver a Minimum Viable Product (MVP) that does the following:

1. Contains the three main parts of the project
    a. Tasker Client
        i. Provides a way to upload Python 2.7 code for execution
        ii. Provides a way to receive results from remote runs of the uploaded code
    b. Server
        i. Provides a way for Tasker Clients to connect and upload Python 2.7 code
        ii. Divides the code into chunks for distibution.
        iii. Connects to Worker Clients to send them Python 2.7 code chunks for execution
        iv. Collects results from the Worker Clients and sends the results back to the correct Tasker Clients
    c. Worker Client
        i. Provides a way to receive tasks (Python 2.7 code chunks) from the Server for execution
        ii. Executes downloaded tasks and records logs
        iii. Sends logs and results back to the server
2. Has a Command Line Interface for each discrete portion

The MVP and the extended product will be delivered to the client — SE 329 Professor (Kent VanderVelden) — and his Teaching Assistant by 14 October 2016.

Maintenance requests for Project Marx can be sent to the project manager Stefan Kraus (sakraus@iastate.edu).

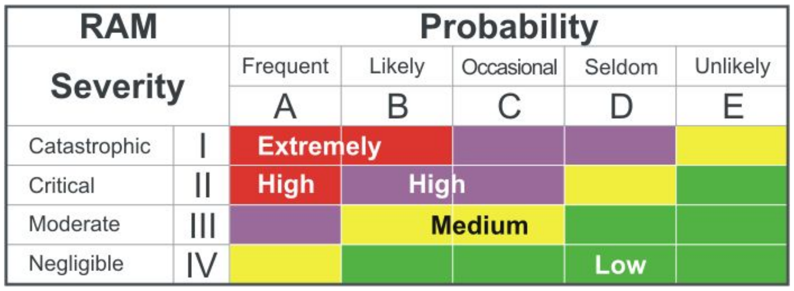
Required from Client

The application will use a dedicated server with Python 2.7 installed as well as a connected network of no fewer than two computers each with the following minimum requirements:

OS: Windows 7 or later, OSX Mountain Lion or later

Processor:	1.3 GHz Intel i3 or similar
RAM:	4GB
Storage:	4GB free hard disk space
Misc:	Internet connection, Python 2.7 installed

Risks



RAM		Probability				
Severity		Frequent	Likely	Occasional	Seldom	Unlikely
		A	B	C	D	E
Catastrophic	I	🟥 Extremely	🟥	🟪	🟪	🟨
Critical	II	🟥 High	🟪 High	🟪	🟨	🟩
Moderate	III	🟪	🟨	🟨 Medium	🟩	🟩
Negligible	IV	🟨	🟩	🟩	🟩 Low	🟩

Figure 2: The Risk Assessment Matrix (RAM) shows the level of risk associated with any given combination of Probability and Severity for a hazard.

Hazard:	Arbitrary code execution
Probability:	Occasional (C)
Severity:	Catastrophic (I)
Assessed Risk:	High
Mitigation:	Encapsulate code running on Worker Machines so that it cannot access the rest of the system.
Assessed Risk After Mitigation:	Low

Hazard: Central Server Crash / shutdown
Probability: Seldom (D)
Severity: Moderate (III)
Assessed Risk: Low
Mitigation: Manual restart of the central server.
Assessed Risk After Mitigation:
Low

Hazard: Tasker Client Crash during task execution, waiting for results
Probability: Occasional (C)
Severity: Critical (II)
Assessed Risk: High
Mitigation: Alternate storage or reporting of the results from a run, or alternatively a kill signal sent to the Worker clients to drop the task, then re-queue upon Tasker reconnection.
Assessed Risk After Mitigation:
Low

Hazard: Unauthenticated use
Probability: Occasional (C)
Severity: Critical (II)
Assessed Risk: High
Mitigation: Secure authentication is a stretch goal as it was not determined to be core to the proof of concept that this project is meant to address.
Assessed Risk After Mitigation:
High (we're likely not mitigating this risk)

Hazard: Worker Machine Shutdown / Disconnection During Execution
Probability: Likely (B)
Severity: Negligible (IV)
Assessed Risk: Low
Mitigation: Retry connection logic, then in the case that the machine cannot reconnect the work will be re-queued.
Assessed Risk After Mitigation:
Low

Hazard: Tasks hanging for long periods of time
Probability: Likely (A)
Severity: Moderate (III)
Assessed Risk: Medium
Mitigation: Add a reasonable, configurable timeout for tasks.
Assessed Risk After Mitigation:
Low

Hazard:	Code failure or Error thrown from code during execution
Probability:	Frequent (A)
Severity:	Negligible (IV)
Assessed Risk:	Medium
Mitigation:	Provide proper error handling and use exit codes to determine how to handle an early or improper exit of a program when running.
Assessed Risk After Mitigation:	Low

Timeline

- **27 September 2016** Project Proposal submitted.
- **10 October 2016** Minimum Viable Product completed.
  - Further testing and extension begins.
- **13 October 2016** Extended or fully tested deliverable completed.
  - Submission time deadline: 11:00 am

Payment Terms

A Minimum Viable Product that fulfills the basic requirements of the assignment should warrant a grade of ‘A’.

Intellectual Property

IJLSA will own and maintain the source code to Project Marx. All product licenses must be purchased through IJLSA and must have the consent of IJLSA to license to others.

License Agreement for the Product
(three-clause BSD License)

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation or other materials provided with the distribution.

Indemnification

The International Justice League of Super Acquaintances will not indemnify Iowa State University for any losses resulting from the project.

Other Thoughts

Beyond the MVP, our stretch goal is to deliver an extended product that has all the functionality of the MVP, as well as some or all of the following:

1. An alternate way to view or receive results, such as an email service or web interface
2. Logic for queueing tasks in the case of more tasks than available worker machines
3. Secure authentication and execution of code sent to the central server
4. Estimated time to completion for tasks based on a benchmark of part of the code
5. Scheduled runs of tasks
6. Basic encryption of network traffic

The alternative way to view or receive results could be either sending completed task results via email to the user who initiated the task or by providing a simple web interface. This could be further expanded by having a history log of task runs that is available to be viewed. It is expected that this may also require secure authentication and authorization.

In order to handle high amounts of task load and make the project more feasible on a large scale operation, it is important to be able to queue tasks until resources are available to service them. A simple solution is to hold tasks in a queue and distribute them when resources become available. This also allows for much greater expansion in “smarter’ allocation of tasks based on expected time to completion on the worker machines, or expected time that the task will take. For example, a one-minute task might be queued before a 30 minute task so that the larger task is only delayed one minute instead of the smaller being delayed 30.

Secure authentication is fairly self explanatory. Users need to be verified in order to keep the system secure and minimize risk of unauthorized use. Having secure encryption would also increase security and allow more sensitive

Estimated time to completion expands on the MVP point of sending a message to the effect of “x / y chunks of code completed” to the task initiator by calculating a time until completion. This information can also be used to improve the queueing logic.