CROWDY
GENERAL-PURPOSE, EXTENSIBLE CROWDSOURCING FRAMEWORK

A THESIS

SUBMITTED TO THE DEPARTMENT OF COMPUTER ENGINEERING

AND THE GRADUATE SCHOOL OF ENGINEERING AND SCIENCE

OF BILKENT UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

MASTER OF SCIENCE

By

Mert Emin Kalender

August, 2012

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Prof. Dr. X (Advisor)

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Assoc. Prof. Dr. Y

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Assist. Prof. Dr. Z

Approved for the Graduate School of Engineering and Science:

Prof. Dr. Levent Onural
Director of the Graduate School

ABSTRACT

CROWDY
GENERAL-PURPOSE, EXTENSIBLE
CROWDSOURCING FRAMEWORK

Mert Emin Kalender
M.S. in Computer Engineering
Supervisor: Prof. Dr. X
August, 2012

Write the abstract here. Please note that an abstract is not the narrative form of contents page. Instead, it should give the same content as in the thesis, and must not be just a pointer to the content of the thesis. Avoid unnecessary words like “In this thesis,”; it is clear that this is the abstract of this thesis, anyway. A good abstract should containe balanced material from each and every chapter of the thesis, and tell essentially the same thing. A common mistake is to write too much from introduction and conclusion chapters, but too little from anything in between.

Keywords: Key one, key two.

ÖZET

TÜRKÇE BAŞLIK

Mert Emin Kalender
Bilgisayar Mühendisliği, Yüksek Lisans
Tez Yöneticisi: Prof. Dr. X
Ağustos, 2012

Türkçe özet buraya gelecek.

Anahtar sözcükler: Anahtar Sözlerim.

Acknowledgement

I acknowledge that …

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The asynchrony, heterogeneity and inherent loose coupling that characterize applications promote system of systems as a natural design abstraction for growing class of software systems. This new concept goes beyond the size of current system definition by several measures such as number of people the system employing for different purposes; number of connections and interdependencies among components; number of hardware elements; amount of data stored, accessed, manipulated, and refined and number of lines of code. Such systems are called Ultra-Large-Scale (ULS) systems and perceived as socio-technical ecosystems [1].

The socio-technical aspect of an ULS system is a result of decentralized and dynamic structure formed by people and software components interacting in complex ways. People become not only users, but also an integral part of the system providing content and computation, and the overall behavior [1]. The difference between the roles concerning system components and humans (user, developer) becomes less distinct. The homogeneity of components ceases together with the increasing scale and variety of people and software components involved within the system. Thus, system gains a social and technical characteristic with the ability to solve numerous problems, even the ones requiring human intelligence.

Social aspect of systems has expanded the scale of collaboration from small or medium sized to internet-scale [2] leading to new era of computation. Collaboration of creative and cognitive people with number-crunching computer systems have appeared under many names such as crowdsourcing, human computation, collective intelligence, social computing, global brain etc, for which you can find detailed studies on classification of systems and ideas in [3, 4] collected under distributed human computation term.

The term *crowdsourcing*, which is the main consideration in this body of work, is first coined by Jeff Howe in the June 2006 issue of Wired magazine [5] as an alternative to the traditional, in-house approaches focusing on assigning tasks to employees in the company for solving problems. Crowdsourcing describes a new mainly web-based business model that exploits collaboration of individuals in a distributed network through an open call for proposals. The term is described by Howe as:

> Simply defined, crowdsourcing represents the act of a company or institution taking a function once performed by employees and outsourcing it to an undefined (and generally large) network of people in the form of an open call. This can take the form of peer-production (when the job is performed collaboratively), but is also often undertaken by sole individuals. The crucial prerequisite is the use of the open call format and the large network of potential laborers. [6]

Crowdsourcing as the new and powerful mechanism of computation has become powerful to accomplish work online [7]. Over the past decade, numerous crowdsourcing systems have appeared on the Web (Threadless, iStockphoto, InnoCentive etc). Such systems enabling excessive collaboration of people have provided the solutions to the problems and tasks that are trivial for humans, but cannot be easily completed by computers or computerized. Hundreds of thousands of people have worked on various tasks including deciphering scanned text (recaptcha.net), discovering new galaxies (galaxyzoo.org), seeking missing people (helpfindjim.com), solving research problems (Innocentive), designing t-shirts (Threadless). Even Wikipedia and Linux can be viewed as crowdsourcing systems from a point of view that conceives crowdsourcing as explicit collaboration of users to build a long-lasting and beneficial artefact [8].

Indeed, one of the most interesting developments in terms of crowdsourcing is Amazon's Mechanical Turk (MTurk), which is a general purpose crowdsourcing platform recruiting large numbers of people to complete diverse jobs. Assignments on MTurk range from labeling images with keywords, transcribing an audio snippet, finding some piece of information on the Web. Requesters submit jobs, which are called Human Intelligent Tasks or HITs in MTurk parlance, as HTML forms. Respectively workers, who are the crowd of users, (called Turkers on MTurk) perform or complete these jobs by inputting their answers receiving a small payment in return. This actual platform and other example systems listed above present the potential to accomplish work in different areas within less time and money required by traditional methods [9, 10].

1.1 Problem

Although current crowdsourcing systems led by MTurk allow a variety of tasks to be completed by people, the tasks requested for completion are typically simple. Tasks, often described as micro-tasks, have the following two fundamental characteristics:

Complexity. Tasks are narrowly focused, low-complex and require little expertise and cognitive effort to complete (taking a couple of seconds to a few minutes).

Dependency. Tasks assigned to humans are independent of each other. The current state of one job has no effect on the other. The result of one job cannot be input to the other to create some information flow.

In that sense, simplicity makes the division and distribution of tasks among individuals easy [11], and independency enables parallelizing and bulk-processing tasks. However, solving more complex and sophisticated problems requires effective and efficient coordination of computation sources (human or software) rather than creating and listing a series of micro-tasks to-be-completed.

Recently detailed analyses on current mechanisms based on foundations of crowdsourcing reveal the necessity of a more sophisticated problem-solving paradigm. Researchers explicitly state the need for a new generic platform with ability to tackle advanced problems. Kittur et al. [12] suggest researchers to form new concepts of crowd work beyond the simple, independent and deskilled tasks. Based on the fact that complex work cannot be accomplished via existing simple and parallel approaches, the authors state requirement for a platform to design multi-stage workflows to complete complex tasks, which can be decomposed into smaller subtasks, by appropriate groups of workers selected through a set of constraints.

In another piece of work, Bernstein et al. [13] regard all the people and computers as constituting a *global brain*, and they indicate the need for powerful new programming metaphors that can more accurately demonstrate the way people and computer work in collaboration. These metaphors are expected to solve dependent sections of more complex problems by decompositions and management of interdependencies. Further, the specification of task sequence and information flow are expected with deliberate collaboration over solutions.

However, recent research only partially addresses these challenges by providing programming frameworks and models [7, 14, 15, 16, 17, 18, 19, 20] for massive parallel human computation, limited-scope and ability user interfaces [10, 21, 22, 23], concepts for planning [24], analysis of collaboration [25]. These works fail to tackle challenges of crowdsourcing due to various reasons: having rigid structure and requirements, being only applicable to a small and bounded problem-set, focusing on a specific aspect of crowdsourcing, being developed in ad hoc manner, requiring a significant amount of work in order to implement and integrate.

Further, human workers are often regarded as homogeneous and interchangeable due to the issues of scalability and availability in existing mechanisms [14]. However, people in a crowd have different skills, and can perform different roles based on their interests and expertise [11]. Current services are created without considering the availability and preferences of people, constraints and relationships, and the support of dynamic collaborations [26]. Thus, human involvement in current mechanisms should be rethought due to limited support for collaboration and ignorance of collaboration patterns in problem-solving [27] over general-purpose infrastructures that can more accurately reflect the collaboration of people and computers [9, 13].

Nevertheless, the development of more generic crowdsourcing platforms along with new applications and structures are expected by the research community [8].

1.2 Purpose

To address the issues, a new extensible general-purpose platform for crowdsourcing is proposed in this work. The new platform, *Crowdy*, is a computing stack that consists of a model-based framework, and on top of that there is the runtime environment supporting design and development of effective and efficient collaboration of human and software components. This platform is concentrated on providing mechanisms that can be used to decompose the implementation of an application into a set of components as a close representation of the real-world problem. The main characteristic of this work is to show how sophisticated problems can be accomplished cleanly and easily relying on component-based model.

1.3 Thesis Plan

The remainder of the work continues as follows. In Chapter 2, background information related to this study is given and previous works are examined. Chapter 3 provides the details of the proposed platform. Motivating examples are described and developed using the proposed platform in Chapter 4. In Chapter 5, a brief discussion of future work is given. A final chapter concludes this study.

Chapter 2

Background

In the following, first the preliminary information about crowdsourcing concepts is summarized, since crowdsourcing is a relatively area of computation. Later, the related studies and works are examined thoroughly.

2.1 Preliminaries

Crowdsourcing term is first proposed by Jeff Howe [5] in 2006. Although there is a long list of terms (collective intelligence, social computing, human computation, global brain etc.), in which some of them are new and some others are old, the use of term ”crowdsourcing” in the academia (demonstrated in Figure 2.1) indicates that research on this domain is promisingly increasing.

Amazon’s Mechanical Turk (MTurk) is the main consideration for the recent academic studies on crowdsourcing. Similar approach is taken in this work to discuss the common terms and form a preliminary list of concepts. MTurk is an intermediary platform between employers (known as requesters) and employees (workers or turkers) for various-sized and difficulty assignments. The platform has become successful for tackling the problems that cannot be solved by computers

[1] The statistics are gathered through ACM Digital Library.

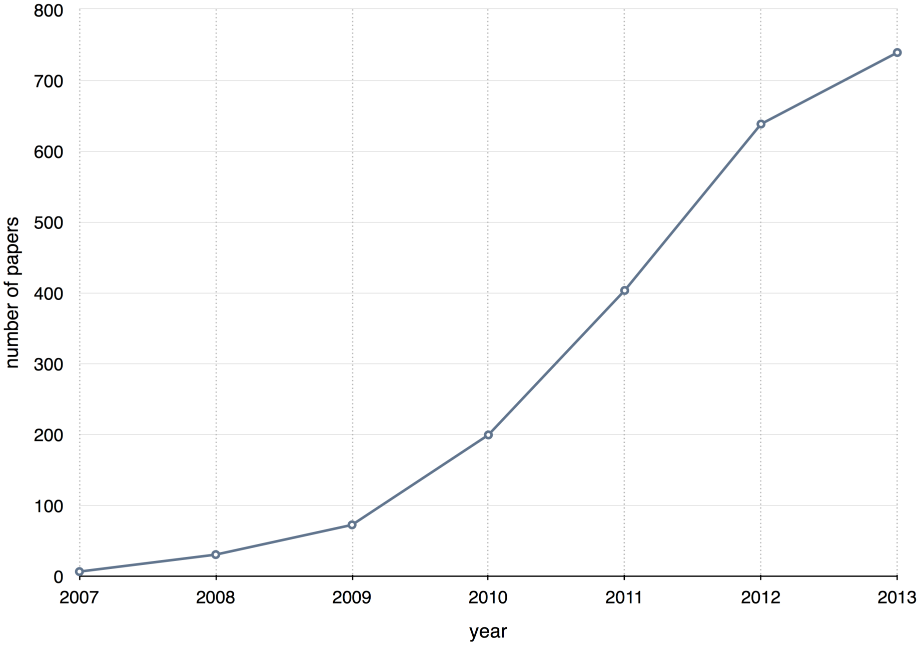

Figure 2.1: The frequency of crowdsourcing keyword in academic papers.[1]

and subject to numerous research studies.

2.1.1 Tasks

Task is a piece of work to be done by the crowd. Terms task, micro-task and job are interchangeably used. The term Human Intelligent Task or HIT is commonly used as well. A task is often expressed over an HTML form in which there are three different input types: single selection (radio buttons), multiple selection (checkboxes) and free text (text areas). For single and multiple selection one or more items can be selected from a list of possible options. Considering free text types workers are supposed to enter a textual response that can be paragraph(s) or sentence(s) or number(s).

In addition to the labels attached to input forms, task has a short description that provides the instructions and keywords, which will help workers search for specific tasks. Further, number of assignments (copies) that requester want completed per task can be set. Additional copies of the same task allows parallel processing. In that case, the system is supposed to ensure that the parallel workers are unique (i.e., no single worker complete the same task more than once).

A task is generally simple requiring small amount of time and expertise to complete. In the following, sample tasks from MTurk are presented.

[Sample Tasks Figure Will Come Here]

Tasks can be grouped into task groups. Task groups are for the tasks that share similar qualities such as tasks to tag images of nature and people or tasks asking for translation of a text snippet from a language to another.

2.1.1.1 Time

Often time scale requiring to complete a task ranges from seconds to minutes. Nearly 20% of tasks takes less than 1-hour, and more than half of tasks does not take more than 16-hours [28]. Maximum time allotted per assignment can be set by requester.

2.1.1.2 Payment

Compensation or reward for completing tasks range from a single penny to dollars. An analysis of MTurk showed that 90% of tasks pays $0.10 or less [28].

2.1.1.3 Acceptance

Once a worker completes and submits an assignment, the requester is notified. The requester has an option to either accept or reject the completed assignment. Acceptance of an assignment, which indicates the work done is satisfactory, makes the worker who completed it get paid, on the other hand rejection withholds payment for which the reasoning may or may not be provided by the requester.

Another option is automatic approval, which is the case when requester does not review work after a some time that can be set by the requester.

2.1.1.4 Expiration

Tasks have a lifetime limit that decides the amount of time that a particular task remains in the listings. Lifetime can be set by requester. After a task reaches end of its lifetime, it is automatically pulled from the listings.

Another type of expiration can happen while a worker is operating on an assignment. When workers accept an assignment, the assignment is reserved for them making no other worker to accept it. The reservation is for particular piece of work (in this case assignment) and time-limited. If worker does not submit the completed assignment in allotted time, then reservation is cancelled and assignment is made available for others again.

2.1.2 Requesters

Requesters are the employers who post tasks, obtain results and pay workers. Requesters are expected to design tasks with all the details (description, instructions, question types, payment, expiration settings etc.) After completed assignments are submitted, requesters can review them by accepting or rejecting, and collect the completed work.

These operations can be done via user interface or application programming interface (API) if there is one.

2.1.3 Workers

Workers are the online users or someone from the crowd who work on and complete assignments in exchange for a small payment. On some platforms (currently not available on MTurk), detailed information about workers are gathered and kept in a database. This information can be later utilized by requesters while associating specific constraints to the assignments such as limiting age to some range for a specific task.

2.1.4 Quality Control

Quality control is challenge for crowdsourcing platforms, since low quality work is common. It is shown that low quality submissions can compromise up to one third of all submissions [21]. As a result, researchers have started to investigate several ways of detecting and correcting for low quality work.

Visualization of workflow is one of the methods employed for which directed graphs are used to show the organization of crowdsourcing tasks, allowing endusers to better understand the problem and proposed solution design [19, 20].

Application of programming paradigms is another approach taken by researchers. MapReduce [7, 14], Divide-and-Conquer [19], Iterative [16], Workflow [15] are some of these approaches taken for organizing and managing complex workflows. Although these paradigms fit perfectly some problems, there are other problems not suited well to those.

Inserting 'gold standard' questions into an assignment with which workers who answer them incorrectly can be filtered out or given feedback [29]. However, writing validation questions create extra burden to requesters and may not be applied to all types of tasks.

Majority voting to identify good submissions is proposed as another option [29, 21], but this technique can be affected by majority (especially when the possible options are constrained) or failed in situations where there are no answers in common such as creative or generative work [23].

Systems (including MTurk) often do not apply any of these quality control approaches, but provide other ways to achieve good workers and discourage bad ones. Currently each worker on MTurk has an acceptance rate that is updated after requester's review on completed assignments. However, this feature does not differentiate one type of task from the other in terms of effect on acceptance rates and that makes it a limited utility. A worker who is skilled in audio transcription would probably have high accuracy rating in related tasks. However, there is no way to reason that this worker can also perform English-to-Turkish translation tasks. Even worse is that workers who pick and complete easy tasks would probably have higher accuracy rates than the ones who choose to perform tasks that require time and expertise [18].

Requesters can utilize acceptance rates by assigning a low limit of them to assignments, which allow only workers with acceptance rates above-limit to accept assignments.

2.2 Related Work

In response to the challenges of crowdsourcing a number of attempts have been made by the researchers. In the following, the studies related to this work are listed, described and examined in detail. The studies that try to address the challenges of crowdsourcing and proposing new way of creating crowdsourcing workflows are considered as related only. Otherwise, the related work list would be too long, since many views Wikipedia and Linux as crowdsourcing systems.

2.2.1 Crowdsourcing Platforms

The following is a list of crowdsourcing studies that propose a new platform to create crowdsourcing services. Each study is discussed extensively and compared to other studies in Table 2.1. The comparison of related works is performed over the building blocks of crowdsourcing platform: concept, existence of human components, existence of software components, requirement for programming, heterogeneity, user interface and progress.

2.2.1.1 Jabberwocky

Jabberwocky [14] is a social computing stack consists of three main components: Dormouse, ManReduce, and Dog. Dormouse is created to enable cross-platform human and machine resource management. It acts like a "virtual machine" layer in the computing stack, consisting of low-level software libraries that interact with people and computers. ManReduce is a parallel programming framework for human and machine computation working on top of Dormouse. ManReduce is inspired by MapReduce [30] that is basically mapping problem into a set of small chunks of work, and then reducing them on an output that aggregate the responses or solutions. Dog is a high-level programming language on top of ManReduce. The language is formed by a small set of primitives for requesting computational work from people of machines.

Jabberwocky is limited in several ways. The computing stack is a command-line tool supported by restricted built-in libraries and can only be run on Dormouse server. The high-level language, Dog, seems simple, clear and expressive, but still there is still a learning curve based on the fact that not all crowdsourcing requesters are developers. This is the issue for the command-line tool as well.

Another important limitation is lack of progress idiom within the system. Jabberwocky receives script definitions (deployment to Dormouse server), a process starts. Once all of them are completed, the output is written to a destination file and process terminates. In that respect, there is no way that end user can observe the current state of problem-solving.

Even if this work aims at solving general-purpose and real-world problems, it is mentioned that real-time and single-worker sequential applications are not well-suited. Despite MapReduce paradigm is simple and many social computing problems fit naturally to this paradigm, it is obvious that only some set of problems are appropriate to be solved using such a paradigm or system, which is another limitation.

However, Jabberwocky's notion of real identity and social structure, which would allow end users to define person-level constraints, is noteworthy.

2.2.1.2 WeFlow

WeFlow [15] is a collaborative application specification, application generator and execution engine proposed in a master thesis [31]. A framework is introduced to create and run collaboration-based applications. That framework allows end users to decompose problem into tasks, describe the computation resources, define the control and data flow.

A task consists of input(s), instruction describing the expected action from user and output(s). Task can be a type of basic (the most simple, atomic work definition), conditional (execution based on a condition), repetition (recurring execution based on a conditional), doall (groups of tasks executed in parallel) or collective (multiple times execution). The framework is definitely restricted by these predefined task definitions.

Besides task specification, end user should designate the control flow and data flow. Control flow determines the order of tasks to-be-executed. On the other hand, data flow is defined to map data between and/or within tasks. All these specifications are done via XML. Despite the fact that the framework is mentioned to require no programming skills and XML is widely used format for representing arbitrary data structures, there is a learning process of WeFlow specifications not to mention the lack of development and debugging environments.

WeFlow differentiates human workers from each other depending on a role definition. Participants are linked to some role. Likewise a task is associated with a role. Although the whole role definition would allow end user handle access control by describing permission-like roles, it does not discriminate human performers based on their characteristics such as age, gender, interest, expertise etc.

2.2.1.3 TurKit

TurKit [16] is a toolkit for deploying iterative tasks to MTurk. Toolkit is based on a model that concentrate on iterative work in which series of individuals work on tasks that are previously completed by others. Although creators of TurKit apply several nontrivial problems (image description, brainstorming, writing tasks, sorting etc.) to the iterative model, this work majorly restricted by the iterative paradigm that toolkit operates on. The complex and sophisticated problems that are expected to be addressed by crowdsourcing systems do not often correspond to iterative model.

TurKit API is defined to help writing iterative MTurk tasks. However, TurKit expects end user to be a programmer and create HTML pages for tasks, and write Javascript files using API. In fact, end user is responsible for testing and making sure that the pages with TurKit functionality interact properly with MTurk. This just reveals another major limitation of TurKit on it's usefulness.

However, the toolkit have some notion of fault tolerance preventing wasted money or time due to bugs and system crashes. Toolkit stores information about the trace of a program's execution. This trace is used whenever program crashes and to put the program back into it's previous state. Thus, toolkit does not re-execute the whole program, but the sections where they are left unfinished.

2.2.1.4 CrowdLang

CrowdLang [17] is a general-purpose framework and a concept of an executable, model-based programming language for workflow definition. The framework is developed based on the assumption that a complex problem is characterized by defining the problem, decomposing the problem into subproblems, planning subproblems, executing the plan and aggregating the solutions of subproblems.

CrowdLang programming language is based on a small set of operators:

Divide-and-Conquer, Aggregate, Multiply, Merge, Router and Reduce (functionality of operators can be understood by their names). These operators are combined together to solve complex problems by routing, distributing and task decomposition. In addition, different types of genes are defined to address various participation patterns.

This work introduces a good and novel concept for general-purpose crowdsourcing that is suitable to most problems. The framework is not bounded to some other programming paradigm or limited to only one aspect of crowdsourcing. It rather supports complex coordination mechanisms. In this respect, translation, which is a sophisticated and difficult problem for crowdsourcing, is attempted in [9] and the results (translation of 15 different articles in less an hour) are promising.

2.2.1.5 AutoMan

AutoMan [18] is described to be a fully automatic crowd programming s ystem. It is a programming system integrating human and computer computation. On top of AutoMan system, a domain specific programming language is defined. It is implemented as embedded domain-specific language for Scala.

The system's whole crowdsourcing concept is based on Question objects. Question can be type of radio-button, checkbox and restricted free-text. The human computation aspect of system only corresponds to the answers given by individuals. Further, system provides no mechanism to design complex problems through a set of subproblems or tasks. This makes system no advantageous to the existing systems depending on simple tasks.

In fact, end users are supposed to write programs in AutoMan DSL and provide them to the system. Considering the examples in the paper, the learning curve for this language can be expected to be high, because it requires knowledge of Scala language (comparing with Dog introduced in [14]).

However, scheduling, pricing and quality control mechanisms are significant components of AutoMan. The runtime system has a scheduling component that periodically checks the current situation of the results, and reprices and restarts human tasks as necessary. By making this, system tries to achieve the predefined confidence level (by end user) while staying under budget.

2.2.1.6 Turkomatic

Turkomatic [19] is a tool that recruits workers for planning and solving complex tasks. The system executes price-divide-solve approach by asking workers to divide complex steps into simpler ones until they are at a simple level, then to solve them. The approach simply uses divide-and-conquer strategy, but the division is done by the crowd different from other crowdsourcing systems.

The system has a set of pre-structured task templates. End users can produce workflows by combining templates together without implementing any software or designing intermediate tasks. The price-divide-solve approach is expected to produce a directed, acyclic task graph in which nodes represent subtasks and links describe task dependencies. The user interface visualizes the task graph and enables endusers delete or modify them in real-time.

The system is significant by demonstrating complex problems through acyclic task graphs. The graphs are not just shown to the endusers, but also implemented in a way that enables real-time modification. However, initial workflow design is left to workers by a single instruction (a few sentences) telling what the enduser achieve in the end. This approach is limited by the efficiency of instruction and understanding of workers, and reportedly not really successful. It is mentioned that for complex work manual intervention and editing of workflow is effective.

Currently the system's crowd planned workflows are not guaranteed to be context free. This restricts the set of problems that the system can tackle, but still this study presents promising results by demonstrating and managing complex problems through acyclic graphs.

2.2.1.7 CrowdForge

CrowdForge [7] is a general-purpose framework and toolkit for accomplishing complex and interdependent tasks using micro-task markets. The framework is built on MapReduce [30] approach, which first breaks down a complex problem into a sequence of subproblems, and combines the results later. A similar approach is taken for the design of ManReduce in Jabberwocky stack [14].

Although CrowdForge approach is designed to fulfill complex problems, it is still limited to the capabilities of MapReduce approach. Some problems may not be addressed by this approach such as the case for tasks that cannot be really decomposed. Another case would be when subtasks are not independent, but the state or result of one is important to complete the other. All these exemplifies the limitations of the approach taken from distributed computing field and expected to fit well in crowdsourcing.

Additionally system has no support for iteration or recursion. It requires the end user (task designer) to specify each stage (partition, map, reduce) in the task flow.

2.2.1.8 CrowdWeaver

CrowdWeaver [20] is a system to visually manage crowd work. This system comes forward among various other works with it's visual representation abilities. High-level representation of a workflow makes it easier for end user to grasp, design and develop crowdsourcing programs.

The system has a predefined set of task templates. Workflows are created by creating tasks and linking them with each other in various ways. Branching and combining multiple data flows are supported and that makes design of complex problems possible.

Besides visualization, CrowdWeaver has a notion of tracking and notification. The task progress component monitors the current state and depicts the current state via graphs. Along with notification component, users can be notified about the progress based on the predefined conditions (specified by users). Further, it is possible to stop a task and make changes on existing branch in real-time.

Despite CrowdWeaver demonstrates a system that can be easily utilized by users with no programming background, the system is currently limited by predefined set of templates. In fact, it's visual abilities are restricted by only showing the workflow, but not enabling users make changes on it directly.

Feature	Jabberwocky	WeFlow	TurKit	CrowdLang	AutoMan	Turkomatic	CrowdForge	CrowdWeaver
Concept	MapReduce	Workflow	Iterative	N/A	N/A	Divide&Conquer	MapReduce	Data Flow
Human components	●	●	●	●	●	●	●	●
Software components	●	●	○	●	●	○	○	●
Requires programming	●	○	●	●	●	○	○	○
Heterogeneity	●	◐	○	○	○	○	○	○
User Interface	○	○	○	○	○	●	●	●
Progress	○	○	○	○	○	○	○	●

●: Yes
◐: Partially
○: No

Table 2.1: Comparison of existing crowdsourcing platforms.

2.2.2 Other Studies

2.2.2.1 Soylent

Soylent [21] is a word processing interface enabling Microsoft Word users to employ MTurk workers to shorten, proofread and edit parts of their documents on demand. The system is developed with respect to the Find-Fix-Verify pattern that is introduced in the same study.

The pattern approaches complex tasks (focused on text editing) by splitting them into a series of generation and review stages. Rather than asking a single worker to read and edit an entire section, the pattern first recruits a set of workers to find areas of improvements. Then, another set of workers review the candidate areas and filter out incorrect ones. Finally, in the verify stage workers perform quality control on previous submissions. Throughout the process the pattern utilizes independent agreement and voting.

Both the system and the pattern concentrates on a small set of use cases of crowdsourcing. The system can only be employed by Microsoft Word users for editing text. The pattern is limited to the problems where decomposition into subproblems is possible.

2.2.2.2 Qurk

Qurk [10, 22] is query processing system that automatically translates queries into tasks to-be-completed by humans. The system has domain-specific language to express tasks. A UDF-like approach is taken by integrating SQL with MTurk expressions.

The approach is concentrated on a MTurk-aware database system rather than a general-purpose crowdsourcing mechanism. Thus, the set of tasks that can be introduced via the system is limited by associated database concepts such as joining, sorting etc. Although generative tasks for which workers give unconstrained input are made available, still processing is done on input data obtained from the underlying database system. However, this study provides an important example by employing people in a database system for managing various queries.

2.2.2.3 Mobi

Mobi [24] is a system to crowdsource itinerary plans. The system illustrates a use case of crowdware paradigm. The paradigm focuses on tasks with global requirements and provides a single workspace in which a crowd of individuals contribute opportunistically to the global solution based on their knowledge and expertise. Itinerary planning is taken as a case study and implemented in Mobi system through a single interface.

In the system and paradigm, the problem definition is limited to the tasks with global constraints. This is a clear example of a study that concentrates on only one aspect of crowdsourcing. However, the study provides insights on the effectiveness of using unified solution context for workers or directing crowd's submissions through a structured guide or benefits of iterative refinements.

2.2.2.4 CrowdSpace

CrowdSpace [23] is a system that supports the evaluation of complex crowd work by combining information about worker results through visualization. This system focuses on exploration and assessment of worker performance and behavior rather than providing ways to manage complex workflows.

The system presents a unified user interface with four components: a scatter plot of aggregate behavioral features (time spent on the task, number of keys pressed while processing the task etc.), distribution of each behavioral features, traces of worker/output pairs and overall worker behavior based on their answers.

Low quality work is common in crowdsourcing. However, this system provides great insights by combining worker behavior with their submissions, and enabling end users to identify the behavior of workers who have good or bad outputs.

Although this approach is limited by several aspects such as being only applicable to pages in which Javascript can be inserted or assuming that worker does all the processing on the page etc, the quality control approach taken in the study can be used to better understand and address the nature of the crowd.

2.2.2.5 Human Architecture

Human Architecture [25] is an adaptation mechanism to the changing requirements of the various type collaborations. Unlike other studies, this work focuses on collaboration problem from the architectural perspective. An architecture description language is proposed to describe collaboration dynamics. Considering software architecture human components and collaboration connectors are introduced to demonstrate coordination dependencies.

However, the proposed human architecture approach is too architecture focused and highly complex. The collaboration of individuals is narrowed down to component and connectors, but in the context of complex problems collaboration is dynamically changing and highly interactive due to the data items that are output from one and input to the other.

Chapter 3

Platform

Crowdy is an extensible general-purpose crowdsourcing platform to solve complex problems. The platform is developed over the fundamentals of stream processing paradigm in which a series of operations are applied to the continuous stream of data elements via operators.

Crowdy is an operator-centric platform. Using this platform, a requester with no requirement of programming background can quickly translate a complex problem into a crowdsourcing application by simply selecting operators and connecting these operators together. Thanks to *Crowdy*'s focus on operators, requesters can design applications by selecting right set of building blocks that are necessary to solve a complex problem and customizing these blocks particular to the computation to-be-conducted.

Crowdy embodies several features:

- A standard toolkit of operators that can employ computation resources (human or software) to accomplish various tasks
- Configuration support to design coordination of computation resources over operators
- Operator parameterization and customization for support of customizable

collaborations

- Application runtime interface

In the remainder of this chapter, the fundamental concepts of *Crowdy* are explained in more detail and features are explored as we look into various aspects of application development using *Crowdy* platform.

3.1 A "Hello World" application

Crowdy applications are developed to solve complex and sophisticated problems that require both human intelligence and computing power. A typical application contains three main high-level components: data ingest, processing and data egress.

Let's consider a minimal "Hello World" application that has these three components in total, connected in a simple pipeline topology. Figure 3.1 demonstrates the application in the form of a flow graph. On the ingest side, there is a *source operator*, which, acts like a data generator. Source operator produces data tuples that are processed down the flow by the *processing operator*. Finally, the *sink operator* simply converts the tuples in such a form (text file, email etc) that can be easily interpreted by requesters.

The application flow graph is specified as a series of operator instances and connections (data flows) that link them. A data flow basically transfers data tuples produced by an operator to another. One or more data segments can be assembled in a data tuple via output specification of an operator instance (see Section'3.3). In addition, several options can be specified to configure an operator instance. These include parameters, operator-specific rules, which is studied in the rest of this chapter.

In a more realistic application, one or more source operators can be employed to produce various data tuples that differ in both size and specification. Similarly,

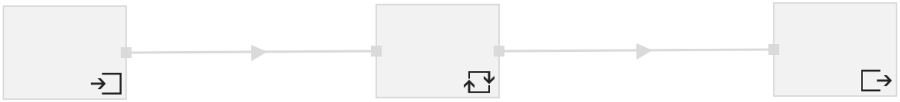

Figure 3.1: A sample, minimal *Crowdy* application.

the application would have one or more processing operators along with other types of operators organized in a way that is significantly more complex than this example.

3.2 Operator

An operator is the basic building block of an application. As soon as an operator is created, a unique ID is generated and assigned to it. Operator has a type that is specified at the time of creation, and type determines configuration respectively. In addition, operator has optional name and description fields that can be used for bookkeeping purposes.

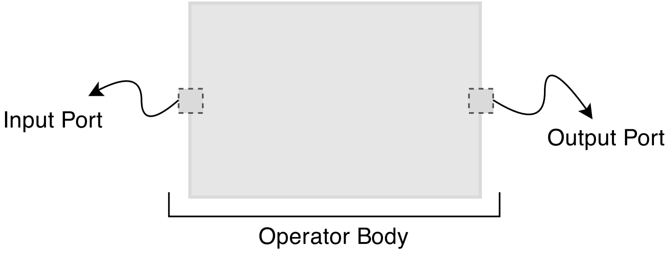

Figure 3.2: Base operator representation.

Crowdy provides a set of built-in operators that can be used to build applications. In general, these operators perform common tasks associated with data generation, processing and outputting. Operators are generally cross-domain to allow general-purpose computation possible.

Operators can be grouped into six main categories: *source, sink, processing, relational, utility, adapter.*

3.2.1 Source operators

The set of source operators generates data tuples. These operators do not have an input port, but have an output port, which outputs data tuples. Figure 3.3 represents a source operator.

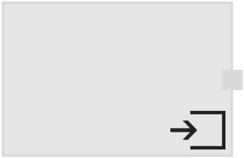

Figure 3.3: Source operator representation.

Operator may have an input port or output port or both corresponding to it's type definition. Figure 3.2 demonstrates a base operator, which consists of a body and ports. Although it is not shown here, each operator presentation has a specific icon on their body associated to it's type.

Source operators together with processing operators are the ones that can be used to specify data flow coming out of an operator. Output specification is an action to identify the data tuple with a series of segments. Other operator types cannot make changes on output specification, but manipulate the flow by dropping or copying data tuples.

human. The *human source operator* is a stateless operator used to produce new data tuples via human workers. Existing crowdsourcing services such as MTurk is used to produce new tuples over the results retrieved from human intelligent tasks. These tasks are automatically created and posted with respect to the specified parameters of the operator.

Human source operator has the following parameters:

- `number of copies`: The maximum number of data tuples can be generated by the operator. It's value ranges from 1 to 1000.
- `max allotted time`: The maximum time in seconds given to a human worker to solve and submit the task. It's value ranges from 10 to 300.

- `payment`: The payment in cents to be given to the human worker in case of successful task completion. It's value ranges from 5 to 500.

- `instructions`: The detailed information for human workers on how to complete the task.

- `question`: The actual sentence asking for specific information.

- `input list`: The list of `input`s that will be shown to human worker to fill.

An `input` can be a type of *text, number, single choice* or *multiple choice.* A text-input presents an input field where the human worker can enter data. The maximum number of characters that can be entered to the field can be set by the requester. Similarly number-input corresponds to a input field where only numbers can be fed in, and the maximum and minimum value for the field are set by the requester. The other two input types conform to input fields where the options given by the requester are presented to the human worker as a list. Human worker is expected to select only one and one or more options for single choice and multiple choice types respectively.

This operator produces a new data tuple for each completed human task. The outputted data tuple consists of data segments corresponding to each `input` specified by requester.

manual. The *manual source operator* is a stateless operator to produce new data tuples.

Manual source operator has the following parameters:

- `manual entry`: The manual text to be parsed and used to produce new tuples.

- `delimiter`: Delimiter to determine segments in a tuple. This can be one of the following values: none, white space, tab, comma, column.

Manual source operator uses manually entered text to create new tuples. Operator retrieves the manual text, parses it line by line and then applies the delimiter. Therefore, each line constitutes a data tuple, and delimiter is used to create segments within a tuple.

[Example]

3.2.2 Sink operators

The set of sink operators is where data tuples are serialized and converted into the formats that can be used by requesters with ease. These operators has one input port, but no output port. Figure 3.4 demonstrates a sink operator.

Figure 3.4: Sink operator representation.

email. The *email sink operator* is a stateless operator to convert data tuples into a text format and email them to requesters. `email` parameter specifies the requester's email address.

file. The *file sink operator* is a stateless operator to serialize the data tuples into a file. Operator has one parameter `filename` that is used to specify the name of file in which tuples will be written.

3.2.3 Processing operators

The set of processing operators provides data tuple processing via human workers. These operators have both input and output ports. Figure 3.5 shows a processing operator.

Figure 3.5: Processing operator representation.

As mentioned before, processing operators can manipulate the data flow specification in addition to source operators. These operators can change the existing flow specification by adding, deleting or editing data segments.

human. The `human processing operator` is almost same as human source operator. The difference is that human processing operator has an input port. That means there is a flow of data tuples coming to the operator. These incoming tuples are made available to requesters via their specification. Therefore, requesters can create an information flow from one operator to another.

The parameters of human processing operator is no different than the parameters of human source operator. However, requesters can use incoming data in `instructions`, `question` and `input list` (applicable to single choice and multiple choice inputs). Available data segments of incoming flow can be placed in these parameters to create human tasks that are handled dynamically.

3.2.4 Relational operators

The set of relational operators enables fundamental manipulation operations on the flow of data tuples. These operators have both input and output port. Each of these operators implements a specific functionality providing continuous and non-blocking processing on tuples.

selection. The *selection operator* is a stateless operator used to filter tuples. A typical selection operator is shown in Figure 3.6.

On a per-tuple basis a boolean predicate is evaluated and a decision is made as to whether to filter the corresponding tuple or not. Boolean predicates are

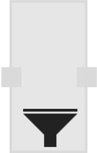

Figure 3.6: Selection operator representation.

specified by requesters as part of operator parametrization. These predicates share the following predefined format:

Filter (in/out) when $\{segment - name\}$ $(equals/notequals/contains)$ $query$

[Explanation]

[Code format]

sort. The *sort operator* is a stateful and windowed operator used to first group tuples and then sort them based on the specified data segment and order. Figure 3.7 illustrates the operator.

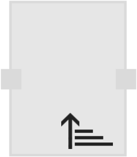

Figure 3.7: Sort operator representation.

Sorting is performed and results are produced (outputted one by one) every time window trigger policy fires. The policy is triggered when the number of data tuples reaches `window size`, which is specified as a parameter.

3.2.5 Utility operators

The set of utility operators provides flow management functions. These operators handle operations such as separating a flow into multiple flows or joining multiple flows into a single one.

enrich. The *enrich operator* is a stateless operator used to enrich data flow by replaying incoming data tuples. Figure 3.8 shows an example view of the operator.

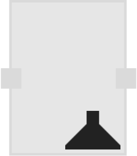

Figure 3.8: Enrich operator representation.

Enrich operator has one parameter called `number of copies`. This parameter determines how many copies will be produced for each incoming tuple.

split. The *split operator* is a stateless operator used to divide an inbound flow into multiple flows. Operator splits the incoming tuples according to the rules specified by requester. When a new tuple comes, the set of rules is evaluated and tuple is transported to the corresponding operator if a rule is triggered, otherwise dropped. Figure 3.9 provides the presentation of a split operator.

[Rule Definition]

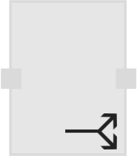

Figure 3.9: Split operator representation.

union. The *union operator* is a stateless operator used to join two or more data flows into one. Different than other operators, this operator can receive more than one flow. These flows are combined by the operator and outputted as if they are a single flow. Figure 3.10 demonstrates the operator.

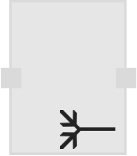

Figure 3.10: Union operator representation.

Union operator requires incoming flows to have the same output specification. Otherwise, union operation will basically fail.

3.3 Flow compositon

TODO

Bibliography

[1] R. P. Gabriel, L. Northrop, D. C. Schmidt, and K. Sullivan, “Ultra-large-scale systems,” in *Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications*, pp. 632–634, ACM, 2006.

[2] C. Dorn and R. N. Taylor, “Analyzing runtime adaptability of collaboration patterns,” in *Collaboration Technologies and Systems (CTS), 2012 International Conference on*, pp. 551–558, IEEE, 2012.

[3] A. J. Quinn and B. B. Bederson, “A taxonomy of distributed human computation,” *Human-Computer Interaction Lab Tech Report, University of Maryland*, 2009.

[4] A. J. Quinn and B. B. Bederson, “Human computation: a survey and taxonomy of a growing field,” in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pp. 1403–1412, ACM, 2011.

[5] J. Howe, “The rise of crowdsourcing,” *Wired magazine*, vol. 14, no. 6, pp. 1–4, 2006.

[6] J. Howe, “Crowdsourcing: A definition,” 2006. [Online].

[7] A. Kittur, B. Smus, S. Khamkar, and R. E. Kraut, “Crowdforge: Crowdsourcing complex work,” in *Proceedings of the 24th annual ACM symposium on User interface software and technology*, pp. 43–52, ACM, 2011.

[8] A. Doan, R. Ramakrishnan, and A. Y. Halevy, "Crowdsourcing systems on the world-wide web," *Communications of the ACM*, vol. 54, no. 4, pp. 86–96, 2011.

[9] P. Minder and A. Bernstein, "How to translate a book within an hour: towards general purpose programmable human computers with crowdlang," in *Proceedings of the 3rd Annual ACM Web Science Conference*, pp. 209–212, ACM, 2012.

[10] A. Marcus, E. Wu, D. Karger, S. Madden, and R. Miller, "Human-powered sorts and joins," *Proceedings of the VLDB Endowment*, vol. 5, no. 1, pp. 13–24, 2011.

[11] H. Zhang, E. Horvitz, R. C. Miller, and D. C. Parkes, "Crowdsourcing general computation," 2011.

[12] A. Kittur, J. V. Nickerson, M. Bernstein, E. Gerber, A. Shaw, J. Zimmerman, M. Lease, and J. Horton, "The future of crowd work," in *Proceedings of the 2013 conference on Computer supported cooperative work*, pp. 1301–1318, ACM, 2013.

[13] A. Bernstein, M. Klein, and T. W. Malone, "Programming the global brain," *Communications of the ACM*, vol. 55, no. 5, pp. 41–43, 2012.

[14] S. Ahmad, A. Battle, Z. Malkani, and S. Kamvar, "The jabberwocky programming environment for structured social computing," in *Proceedings of the 24th annual ACM symposium on User interface software and technology*, pp. 53–64, ACM, 2011.

[15] N. Kokciyan, S. Uskudarli, and T. Dinesh, "User generated human computation applications," in *Privacy, Security, Risk and Trust (PASSAT), 2012 International Conference on and 2012 International Confernece on Social Computing (SocialCom)*, pp. 593–598, IEEE, 2012.

[16] G. Little, L. B. Chilton, M. Goldman, and R. C. Miller, "Turkit: tools for iterative tasks on mechanical turk," in *Proceedings of the ACM SIGKDD workshop on human computation*, pp. 29–30, ACM, 2009.

[17] P. Minder and A. Bernstein, "Crowdlang-first steps towards programmable human computers for general computation.," in *Human Computation*, 2011.

[18] D. W. Barowy, C. Curtsinger, E. D. Berger, and A. McGregor, "Automan: A platform for integrating human-based and digital computation," in *ACM SIGPLAN Notices*, vol. 47, pp. 639–654, ACM, 2012.

[19] A. Kulkarni, M. Can, and B. Hartmann, "Collaboratively crowdsourcing workflows with turkomatic," in *Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work*, pp. 1003–1012, ACM, 2012.

[20] A. Kittur, S. Khamkar, P. André, and R. Kraut, "Crowdweaver: visually managing complex crowd work," in *Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work*, pp. 1033–1036, ACM, 2012.

[21] M. S. Bernstein, G. Little, R. C. Miller, B. Hartmann, M. S. Ackerman, D. R. Karger, D. Crowell, and K. Panovich, "Soylent: a word processor with a crowd inside," in *Proceedings of the 23nd annual ACM symposium on User interface software and technology*, pp. 313–322, ACM, 2010.

[22] A. Marcus, E. Wu, D. R. Karger, S. R. Madden, and R. C. Miller, "Crowdsourced databases: Query processing with people," CIDR, 2011.

[23] J. Rzeszotarski and A. Kittur, "Crowdscape: interactively visualizing user behavior and output," in *Proceedings of the 25th annual ACM symposium on User interface software and technology*, pp. 55–62, ACM, 2012.

[24] H. Zhang, E. Law, R. Miller, K. Gajos, D. Parkes, and E. Horvitz, "Human computation tasks with global constraints," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pp. 217–226, ACM, 2012.

[25] C. Dorn and R. N. Taylor, "Architecture-driven modeling of adaptive collaboration structures in large-scale social web applications," in *Web Information Systems Engineering-WISE 2012*, pp. 143–156, Springer, 2012.

[26] D. Schall, S. Dustdar, and M. B. Blake, "Programming human and software-based web services," *Computer*, pp. 82–85, 2010.

[27] C. Dorn, R. N. Taylor, and S. Dustdar, “Flexible social workflows: Collaborations as human architecture,” *Internet Computing, IEEE*, vol. 16, no. 2, pp. 72–77, 2012.

[28] P. G. Ipeirotis, “Analyzing the amazon mechanical turk marketplace,” *XRDS: Crossroads, The ACM Magazine for Students*, vol. 17, no. 2, pp. 16–21, 2010.

[29] C. Callison-Burch, “Fast, cheap, and creative: evaluating translation quality using amazon’s mechanical turk,” in *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 1-Volume 1*, pp. 286–295, Association for Computational Linguistics, 2009.

[30] J. Dean and S. Ghemawat, “Mapreduce: simplified data processing on large clusters,” *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[31] N. Kokciyan, “Weflow: We follow the flow,” Master’s thesis, Galatasaray University, 2009.