Vol.2021-CE-159 No.24
2021/3/14

リアルタイムでモニタする並べ替えプログラミング・パズルにおけるユーザ操作の傾向と性能設計

山口 琢[1]　松澤 芳昭[2]　大場 みち子[3]

概要：プログラミング学習・指導の支援システムが、実験だけでなく授業でも使われる事例は、今後増えるだろう。これらの中には事後だけでなく、リアルタイムでも受講生の様子をモニターするものもある。並べ替え型のプログラミング・パズル・システムのジグソー・コードを大学 1 年生約 230 名の実習授業で使ったとき、問題ページへのアクセスの秒間最大リクエスト数が 33 件/秒に達し、問題が表示されるまで 2 分以上待たされるという問題が起きた。また、このとき、操作ログのデータの一部が失われた。対策として、急なアクセス増に対応するため、システムを複数のサービスに分割し、スケーリングのパラメタを調整した。また、操作ログの送信をリトライするように修正した。問題を再現できるテスト環境を構築し、これら対策の効果を確かめた。このようなシステムでは、操作ログに限らず、ユーザ操作による再読み込みを含めて、リトライを前提にシステムを設計する必要があると考える。今後は、ログ送信にリトライを導入したことによって操作ログが重複して登録される可能性が生じたので、分析にあたって考慮する必要がある。

The Tendency of User Operations and The Performance Design in A Reassembling Programming Puzzle Which Monitors The Users' Operations in Real-Time

Taku Yamaguchi[1]　Yoshiaki Matsuzawa[2]　Michiko Oba[3]

1. はじめに

プログラミング教育において，学習者の演習プロセスを測定・分析して，学習者のつまづきを検出したり理解度を測ったりする研究が行われるようになってきた [1][2][3][4][5][6]。これらの中には事後だけでなく、リアルタイムでも受講生の様子をモニターするものがある。

このようなシステムが、実験だけでなく授業でも使われる事例は今後増えるだろう。大学入試でも CBT (Computer Based Testing) の導入が検討されている [7]。そのような実システムを開発・運用するためには、学習・指導システムが受ける負荷についての定量的な見積もりと、それに応じたシステム設計が必要である。

本稿では、われわれが研究・開発・運用している並べ替えプログラミング・パズル・システムで起きた性能と操作ログ取得の問題を調査して、主にユーザからのアクセス状況を分析し、併せてシステム側の対策を報告する。問題が発生した授業では、受講生によるアプリケーション操作から受講生の思考過程を分析する研究者がいるが、本稿はその研究者の立場ではなく、アプリケーションを開発・提供した研究者の立場でのものである。

以下、2 章では先行研究を調べる。3 章では、報告の対象であるジグソー・コードシステムについて、本稿の観点から説明する。4 章では、発生した問題を調査して 3 つの課題に整理する。5 章、6 章および 7 章ではそれぞれの課題について、これまでの経緯や対策や考察を述べる。8 章では対策結果を確認する。9 章では全体を考察する。10 章

[1] フリー
Independent Researcher
[2] 青山学院大学　社会情報学部 社会情報学科
School of Social Informatics, Aoyama Gakuin University
[3] 公立はこだて未来大学システム情報科学部
Faculty of Systems Information Science, Future University Hakodate

でまとめと、残された課題を述べる。

2. 先行研究

学習アプリケーションを開発して、ユーザ操作を測定・分析する学習分析の研究は数多く報告されるようになった。しかし、学習に直接関係しないためか、システムへの同時アクセス数やレイテンシ (laytency、待ち時間、「待たせる時間」) を分析して設計を評価する報告、あるいは分析対象における欠損データの有無の報告は少ない。

文献 [7] は、大学入試を想定した CBT システムのプロトタイプを開発して、試験問題を作成して CBT を実施した研究の報告である。システムは Web アプリケーションとして実装され、受験生は Web ブラウザからサーバにアクセスした。この研究の主な対象は試験問題である。1,406 名の高校生が受験したとの報告がある (p.50) が、サーバへのリクエスト数やレイテンシに関する定量的な報告はない。

また、文献 [7] の「別添資料 6 大規模 CBT システム構築への課題とその解決策」は、大規模 CBT を検討したものである。入試用の専用端末を導入し、入試用アプリケーションまたは Web ブラウザを使い (p.4)、試験問題は事前に端末に配付し (p.3)、受験者の解答も会場内ネットワークの分散ハッシュテーブル (Distributed Hash Table、DHT) に時々刻々と保存する (p.6) ことを提案している。

文献 [7] の別添資料 6 には、この他にも参考になる記述がある:「ゲームのプラットフォームの実例…プレイ中の利用者も 250 万人居る。」「ゲームは有料なので、…遅延に対しては、厳しいと予想される」(p.12)、「(仮想サーバより物理サーバが無難であるが) 年 1 回の試験では、物理サーバを確保するのはコスト的に難しいのではないか」(p.13)。「合格発表に Web をりようしている大学では、Web サーバ 1 台あたり 700 同時アクセスを設計上の上限としている」(p.14)。

3. これまでの取り組み

本稿の対象システムである、並べ替えアプリケーション「ジグソー・コード」の機能と開発経緯、および本稿でのポイントとなる要件の概要を述べる。

3.1 並べ替えアプリケーションと操作の測定

ジグソー・コードは並べ替えプログラミングのアプリケーション (**図 1**)、ジグソー・テキストは並べ替え作文のアプリケーションである。プログラミングやテクニカルライティングの演習で使われている。

これらアプリケーションは、ユーザの並べ替え操作を測定している (**図 2**)。われわれは、これらアプリケーションの機能/ユーザ・インタフェース (UI)、操作ログのデータ、データの分析手法、分析結果を学習・指導に活用する手法を研究してきた [2][4]。

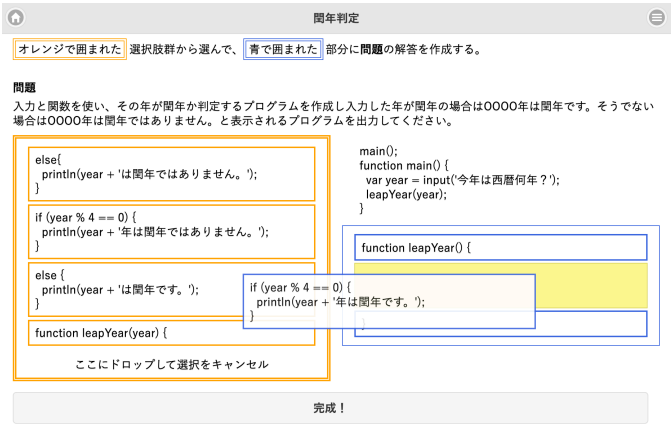

図 1 並べ替えプログラミング・パズルのジグソー・コード

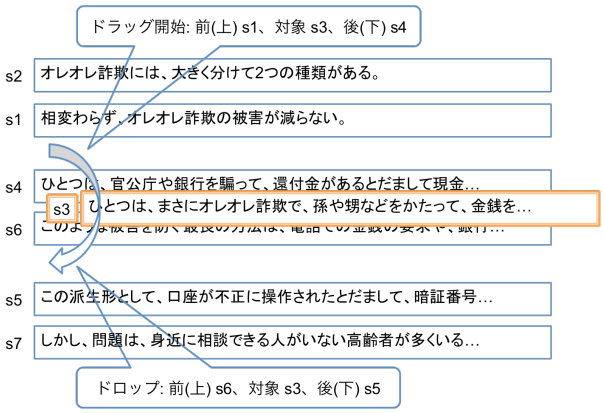

図 2 並べ替え作文のジグソー・テキストにおける、並べ替え操作の測定

3.2 ネイティブアプリでなく Web アプリ

これらのシステムは、iPhone といったデバイスのネイティブ・アプリケーションではなく、Web アプリケーションとして実装ている。

これは、幅広いデバイスに対応しつつ、開発量を抑えて開発期間を短くするためである。Web アプリケーションとすることで、学生や社会人を対象としたアドホックな講習会/ハンズオンなどでも利用しやすい。実験でさえ、被験者のデバイスにアプリをインストールしてもらうところから始めるのは敷居が高い。

Web ブラウザでは標準技術の採用が進み、1 つの実装で、Windows や Mac といったパソコンから iPhone や Android などのスマホでも利用できるようになった。実際、文献 [8] の実践では、受講生の全員が個人のスマートフォンで授業を受け、OS は iOS と Android の 2 種、機種も多岐にわたった。

3.3 処理の流れ

本稿のシステムは、Google 社のクラウド・コンピューティング・サービスの 1 つであり、PaaS (Platform as a Service) である Google App Engine を使っている。サーバ側の処理は Python 2 と Python 3 で記述している。

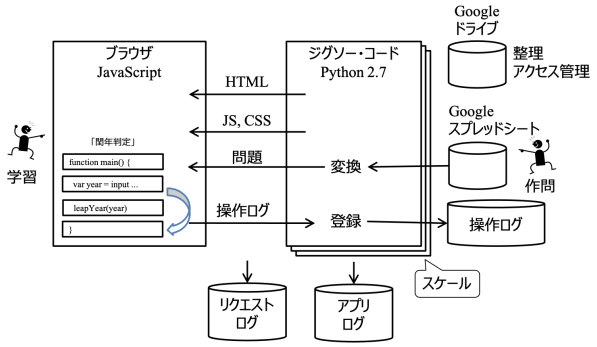

図 3 ジグソー・コードとジグソー・テキストの構成

図 3 は、4 章で述べる問題発生時の図 1 や図 2 の、アプリケーションの処理を説明している。

図 1 の画面は、HTML と CSS と JavaScript で記述されている。ブラウザが図 1 のような問題ページをサーバにリクエストすると、ページを記述した HTML がサーバから返される。その HTML を解釈して、ブラウザは CSS や JavaScript のスクリプト・ファイルをサーバにリクエストする。読み込まれた JavaScript は、並べ替え操作を実現するとともに、問題データをサーバに Ajax でリクエストする。問題データは Google スプレッドシートで記述されている。サーバは Google スプレッドシートの API(Sheets API) を使って問題データを取り出し、変換してブラウザに返す。この問題データのリクエストに対する処理は Python 2.7 で記述されている。ブラウザ側の JavaScript が、このデータを解釈して問題が表示される。

図 2 のユーザ操作の測定処理は JavaScript で記述されている。ユーザがプログラム行をドロップなどすると、ドロップされた行の ID や操作時刻などを、JavaScript が Ajax でサーバに送信する。サーバは、操作ログ・データを受け取って、データベースに登録し、登録処理の成否をブラウザに返す。

ジグソー・コードのサーバ側のプログラムは、負荷に応じて自動的にスケールする。

3.4 教師によるリアルタイム・モニタリング

本システムは、図 2 で捉えたユーザの操作を、操作のたびにサーバに送信していて、この測定データをリアルタイムに分析できる。これによって、学生が解く時々刻々の様子を把握し、問題を解く考え方・解き方をその場で指導できるよう目指している。

実際、受講生が予想したパターンで行を動かしているか、何人がどこまで進んだか、完成したかなどが分かる。

3.5 スパイラルな研究・開発

ユーザ操作を測定・分析する我々の研究は、約 10 年前からスパイラルに進めてきた。最初のシステムはワークシート作文で 2011 年から、ユーザ操作の測定・分析は 2012 年頃から取り組んできた。これらは最初から Web アプリケーションで、操作ログを送信する JavaScript のコードや、操作ログ・データを受け取って保存するサーバ側の Python 2.7 のコードは、現在のジグソー・コードなどに引き継がれている。

アプリケーションの機能/UI、操作ログのデータ、データをリアルタイムあるいは事後に分析する手法、分析結果を学習・指導に役立てる手法は、相互に関連している。授業に必要な機能や分析に必要なデータを後から追加したり、後に活用されなくなったデータ項目がデータベースに残ったり、活用されなくなった仕組みが処理方式に残っていたりする。

4. 問題発生

2020 年度後期の授業でジグソー・コードを使ったとき、問題が発生した。

授業は、青山学院大学の文理融合型の学部 1 年生向け必須授業のコンピューティング実習。内容は、JavaScript、HTML を使って、変数、条件分岐、繰り返し、関数などプログラミングの初歩を学ぶ。受講したのは約 230 人。授業はオンライン形式であった。次の 4.1 節で述べる通り、サーバ側に記録された IP アドレスが 234 通りあったことから、ユーザは自宅などからアクセスしていたと考えられる。

授業中に小テストとして、ジグソー・コードの問題を解いた。解いた問題は 3 問で、1 問目はタートルを使ってドーナツを描くプログラムを完成させる「ドーナツプログラム」、2 問目は入力した年が閏年かを判定する「閏年判定」、3 問目はタートルを使って「手裏剣を描くプログラム」であった。「始め」の合図で一斉に解き始め、2 問目以降を解き始めるタイミングは任意であった。

複数回の授業の小テストで、問題を解き始められるようになるまで 2 分以上待たされるという問題が起きた。

また、ある回の授業について、サーバに対するリクエストのログを調べたところ、操作ログのデータの一部が、データベースへの登録に失敗し、データが失われていたことが分かった。

サーバへのリクエストのログには、リクエストの URL が記録されている。この URL によって、そのリクエストが問題の HTML を求めたものか、操作ログの登録を求めたものかが分かる。しかし、例えば POST リクエストの本文 (フォームの中身) などは残らないため、操作ログの内容などは分からない。

以下、サーバへのリクエストのログをリクエスト・ログ、サーバ・アプリケーションが出力したログをアプリケーション・ログ、ユーザ操作のログを操作ログと呼び分ける。リクエスト・ログ、アプリケーション・ログ、およびデータベース登録に成功した操作ログから、分かる範囲で現象を調査する。

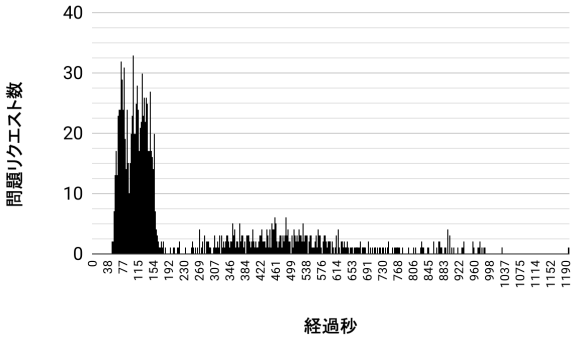

図 4　3 問を合算したサーバへのリクエスト数

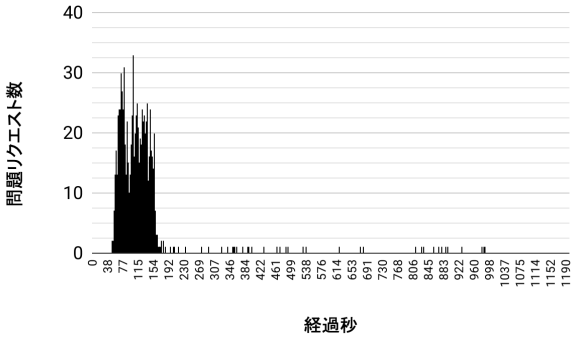

図 5　「ドーナツプログラム」へのリクエスト数

表 1　問題ごとのサーバ側のレイテンシ: 平均値、中央値、最大値、最小値

問題	平均 (秒)	中央	最大	最小
ドーナツプログラム	13.818	10.249	42.111	0.007
閏年判定	2.956	0.012	31.535	0.007
手裏剣を描くプログラム	2.736	0.011	41.317	0.006
全体	10.568	9.450	42.111	0.006

4.1　ユーザ操作の分析

リクエスト・ログから、ユーザの操作を推定する。

4.1.1　リクエスト数とレイテンシ

図 4 は 1 秒間のリクエスト数の推移で、縦軸は 3 問を合算したリクエスト件数、横軸が経過時刻で 1,200 秒 = 20 分間にわたる。問題ページへのアクセスが急激に立ち上がり、アクセス開始から 52 秒かけて秒間最大リクエスト数が 33 件/秒に達していた。この時点までの累計リクエスト数は、206 個の IP アドレスから 775 件であった。また、**図 5** から**図 7** は、これを問題ごとに分離したもので、図 5 は 1 問目「ドーナツプログラム」への、**図 6** は 2 問目「閏年判定」への、図 7 は 3 問目「手裏剣を描くプログラム」への、1 秒間のリクエスト数の推移である。1 問目に比べて、2 問目や 3 問目は立ち上がりが緩やかで、秒間リクエスト数も少ない。1 問目は「始め」の合図で一斉に解き始めるが、2 問目以降は解き始める時間がばらけることが、データに現れている。

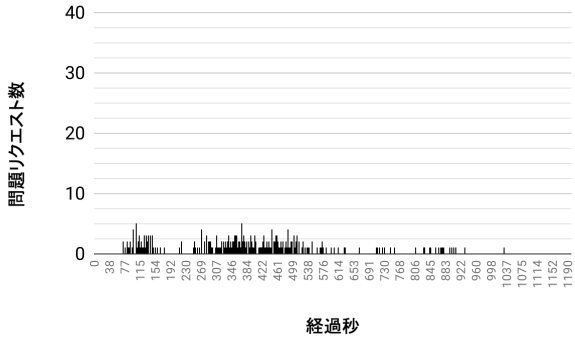

図 6　「閏年判定」へのリクエスト数

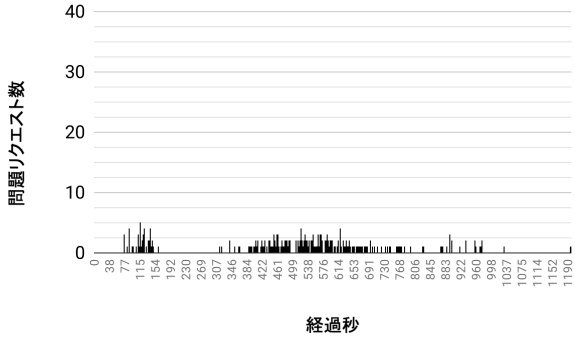

図 7　「手裏剣を描くプログラム」へのリクエスト数

表 1 は問題ページごとのサーバのレイテンシの、平均値、中央値、最大値、最小値である。これは問題ページの HTML のリクエストをサーバが受けてから、HTML データを返すまでの時間である。ユーザが問題を解き始められるまでには、これに加えて JavaScript や CSS が読み込まれ、さらに問題データが読み込まれるのを待つ必要がある。ブラウザ側でユーザが経験した待ち時間のデータはないが、負荷の少ない通常は、問題を解けるようになるまで数百ミリ秒程度かかる。

表 1 では、1 問目の「ドーナツプログラム」が平均と中央値のどちらも 10 秒を超えているのに対し、2 問目以降は中央値で 0.01 秒程度であり、開始直後に応答が遅かったことが分かる。2 問目と 3 問目の平均値が中央値に比べて大きいのは、レイテンシの遅かった最初の時間帯に、1 問目を諦めて 2 問目や 3 問目を読み込む操作をした場合があったためと考えられる。

問題ごとのリクエスト件数の合計は**表 2** のようになっていて、1 問目の「ドーナツプログラム」へのリクエストは 1,607 件であった。受講生が約 230 人なので、1 人あたり約 7 回リクエストしたことになる。これは、応答が遅いので、ブラウザで再読み込みしたためと考えられる。

ブラウザ側の再読み込み操作のデータはないが、IP アドレスごとのリクエスト数は、リクエスト・ログから分かる。**図 8** は IP アドレスごとのリクエスト数、**図 9** はリクエスト数の度数分布である。3 問の問題に対して、リクエスト

表 2　問題別のリクエスト数

問題	リクエスト件数
ドーナツプログラム	1,607
閏年判定	355
手裏剣を描くプログラム	322
(その他の問題)	3
合計件数	2,287

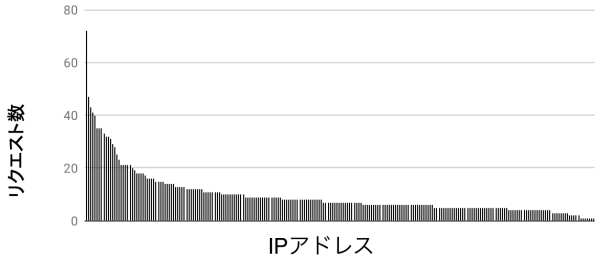

図 8　IP アドレスごとのリクエスト数

数の平均値は 9.8 回、中央値は 7 回、最大値は 72 回、最小値は 1 回であった。

4.1.2　ユーザの区別とオンライン授業

一般的には IP アドレスとユーザとが 1 対 1 に対応してるとは限らないが (図 **10**)、ユニークな IP アドレスが 234 個記録されていたので、ほぼ 1 対 1 であったと考えられる。例えば**表 3** は、ある特定の IP アドレスからのリクエスト履歴であるが、問題 1「ドーナツプログラム」を複数回リクエストしている一方で、問題 2 と問題 3 には 1 回しかリクエストしてないので、1 人のユーザと考えられる。最大値 72 回を記録した IP アドレスについて同様に調べると、各問題を複数回リクエストしているので、複数のユーザが対応してると考えられる。問題が 3 問あるのに対してリクエスト数の最小値が 1 回では数が合わないが、これら IP アドレスには教師や TA(Teaching Assistant) などが含まれている可能性がある。

IP アドレスとユーザがほぼ 1 対 1 であったのは、授業がオンライン授業であった今年度の状況を反映しているだろう。全員が同じローカルネットワークからインターネット上のサーバにアクセスする状況では、こうはならなかったと考えられる。

4.1.3　リクエスト間隔と再読み込み

表 3 は、合計 20 回リクエストした IP アドレスからの、リクエストの経過である。表の上から下へ時間が経過し、各行が各リクエストに対応する。前回のリクエストから 0.2 秒でリクエストしている場合は、ブラウザの応答を待って操作していないと考えられる。以前の授業で待たされた経験から、この回の授業ではこのような操作になっているか、あるいは、これが「デジタル・ネイティブ世代におけるアプリ操作の作法」である可能性もある。

なお、表 3 で分かるように、ユーザがブラウザで再読み込みによって新たなリクエストをしても、サーバは前のリクエストの処理を続けている。例えば、リクエストの 0.2 秒後に再読み込みしても、サーバは 5.9 秒かけて再読み込み前のリクエストを処理して応答している。その間にサーバ資源を消費しているので、サーバ側内で、ある処理が別の処理の応答を待ち続けるべきかどうかが、設計上のポイントの 1 つとなることが分かる。

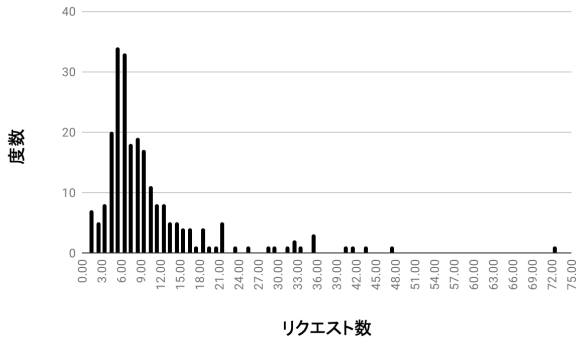

図 9　IP アドレスごとのリクエスト数の度数分布

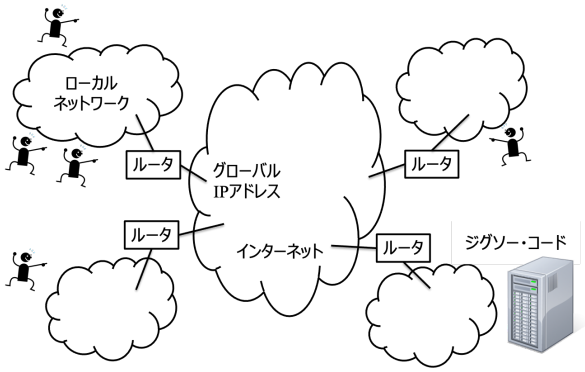

図 10　IP アドレスとユーザの区別

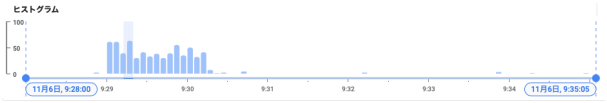

図 11　サーバ側のエラー件数の推移、縦軸が件数、横軸が時間。

4.2　サーバ処理の分析

リクエスト・ログから、問題へのリクエストが始まったのは午前 9 時 28 分 50 秒で、28 分 53 秒に問題データ取得処理でエラーが発生し、その後、2 分ほどの間にエラーが頻発していたことが分かる (図 **11**)。問題が表示されるまで 2 分以上待たされたという現象を、リクエスト・ログからも確認できた。

エラーの内訳を細かく分析すると、まず解答開始時に、問題データを Google スプレッドシートから取り出す処理が遅くてアプリケーション内でタイムアウトしていた (152 件)、または処理が待たされたあげくアプリケーションに渡されることなく中断 (abort) していた (21 件)。9 時 30 分ころには、4.1.1 項で分析した問題の HTML へのリクエス

表 3　あるユーザからのリクエストの間隔 (秒) と、各リクエストごとのサーバ側レイテンシ

問題	リク間隔 (秒)	レイテンシ
ドーナツプログラム	-	33.9
ドーナツプログラム	4.5	23.4
ドーナツプログラム	6.4	27.7
ドーナツプログラム	0.7	27.1
ドーナツプログラム	0.2	5.9
ドーナツプログラム	0.2	26.7
ドーナツプログラム	0.5	26.1
ドーナツプログラム	0.2	26.0
ドーナツプログラム	0.2	25.7
ドーナツプログラム	0.3	13.9
ドーナツプログラム	9.4	10.1
ドーナツプログラム	4.7	30.9
ドーナツプログラム	7.6	30.6
ドーナツプログラム	43.0	3.6
ドーナツプログラム	2.3	11.3
ドーナツプログラム	3.4	3.4
ドーナツプログラム	0.9	4.9
ドーナツプログラム	0.1	1.8
閏年判定	373.5	0.0
手裏剣を描くプログラム	207.8	0.0

表 4　問題のプレイ数、操作数 (TBD)

問題	プレイ数	全操作	操作/プレイ
ドーナツプログラム	306	7,275	23.8
閏年判定	274	5,719	20.9
手裏剣を描くプログラム	261	9,126	35.0
合計	841	22,120	

トそのものが、30 秒以上待たされたあげくアプリケーションに渡されることなく中断し始めた (378 件)。また、問題を表示できたユーザが問題を解き始めたが、その操作ログを保存する処理が、同様に中断していた (15 件)。

アプリケーションに処理が渡されることなくリクエストが中断された場合、アプリケーションとしては、それを例外 (Exception) として捉えることも、クッキーなどでユーザ操作を追跡することもできない。

4.3　操作ログと欠損の評価

4.3.1　プレイ数

本システムの研究では、問題を解くことをプレイすると呼んでいる。1 つの問題を 1 人のユーザ (プレイヤー) が何度も解くことがあるので、授業中のプレイ数は問題を解いた延べ人数となる。授業時間の午前 9 時から 12 時の間に、ジグソー・コードの 3 つ問題を解いたプレイ数を**表 4** に整理する。

受講生が約 230 人なのに対して、最初の問題「ドーナツプログラム」のプレイ数が 306 件と多い。これは、教師や TA の存在を想定しても、かなり多い。この原因は、システムの反応が遅いために何度も再読み込みしたことが反映

n \ n+1	s3	s4	s5	s6	s8	s9	s10	s11
s3	29	32	14	35	13	27	15	18
s4	18	48	76	21	5	66	32	36
s5	31	27	34	76	15	18	22	25
s6	39	15	11	38	25	13	10	11
s8	15	12	6	21	27	6	3	5
s9	10	77	32	13	5	38	38	35
s10	17	14	24	28	9	17	39	23
s11	25	14	42	35	10	10	16	47

図 12　並べ替え操作の共起行列

していると考えられる。ただし、プレイが 1 件として記録されるのは、問題データがブラウザに読み込まれてプレイ可能 (解き始められる) となってからである。解き始められるようになったにもかかわらず再読み込みしたか、あるいは、最初の問題を何度か解き直したかであろう。

4.3.2　操作ログの全件数と失敗数

授業時間の午前 9 時から 12 時までの間に、データベースへの登録をリクエストとされた操作ログの件数は 22,104 件であった。このうち 15 件のリクエストが、アプリケーションに渡されず中断された。

表 2 から所定の 3 問以外の問題も 3 つ解かれた (プレイされた) ことが分かる。この所定外の 3 問を無視して、22,104 件を表 4 の合計プレイ数で割ると、1 問あたり平均 26 操作で解いていることが分かる。

4.3.3　欠損の評価

操作ログの登録リクエストのうち登録できなかったのは、6 個の IP アドレスから合計 15 件であった。これは、合計リクエスト数 22,104 件の 0.07%にあたる。これが多いか少ないか、簡単に評価する。

ジグソー・コードの操作ログの分析では、操作過程のパターンを見つけるために、操作対象となった行 (パズルのピース) の共起行列を使うことが多い。共起行列は、ある行を動かした次にどの行を動かしたかの回数を集計する。**図 12** は、データベースに登録された「ドーナツプログラム」の操作ログを集計した共起行列である。行や列の見出しにある s4 や s5 は「ドーナツプログラム」に含まれる行 (ピース) の ID である。図では、s4 行の次に s5 行を動かした回数が全部で 76 あったことが分かる。

図のように共起頻度には偏りがあり、操作の傾向を表していると考える。「s4 の次に s5」という共起の頻度の評価として、共起頻度の平均値と標準偏差を使う。図では、セルの値の合計が 1,608、平均値が 25.1、標準偏差が 16.9 である。平均値 + 標準偏差 = 42 よりも大きいセルを黄色の背景 (薄い背景)、平均値 + 標準偏差 × 2 = 58.9 より大きいセルを赤い背景 (濃い背景) で示す。

仮に、失われた操作ログ 15 件が全て s4 の次に s4 を動かすものだったとすると、「s4 の次に s4」という共起の頻度は 48 + 15 = 63 となり、赤い背景に該当することになる。しかし、そのような確率は 48 / 1,608 = 3%であることを共起行列は示している。さらに、操作ログには種類があって、「プレイ開始」、「プレイ完了」、「行をドラッグ開始」、「行をドロップ」、「行を (左右どちらかの) 枠から取り出す」、「行を (左右どちらかの) 枠に入れる」の 6 通りである。共起行列は「行をドロップ」だけを集計する。「行をドラッグ開始」と「行をドロップ」は必ず対で発生するので、失われた操作ログの半分が「行をドロップ」であると見積もると、3%よりもさらに小さくなる。ただし、15 / 22,104 = 0.07%程度に小さいとまでは言えない。

結局、本件については、失われた操作ログが共起行列に基づく分析に影響を与えた可能性は小さいと考えられるが、直ちに無視できるとはいい難い。

4.4 課題の整理

以上の分析より、問題は 3 つに整理できる:

(1) まず問題データを Google スプレッドシートから取り出す処理が重い
(2) さらにこの処理の重さによって他の処理が実行されず中断してしまう
(3) 研究の観点からは、操作ログデータが失われることが問題である。

以下、パズル問題データの取得処理の改善、システム構成の再構築、操作ログ送信のリトライの 3 課題に分けて、問題点と対策内容を説明する。

5. 課題 1: パズル問題の取得処理の改善

5.1 検討と開発の経緯

問題は Google スプレッドシートで記述され、Google ドライブによってフォルダに整理され、アクセス管理されている。この方法で実現している理由は、一定の使い勝手を満たしつつ、開発のコストを抑えて短時間・高品質で実装するためである。

問題の作成者はシステム開発者ではないし、ジグソー・テキストによる並べ替え作文の場合にはプログラミングの素養を前提にできない。問題データを、引用符「"」などを気にしながら CSV で記述するより、スプレッドシートに記入するほうが、問題作成の敷居が低い。Google スプレッドシートの、アプリケーションとしての操作性も十分であると考えた。

ジグソー・コードやジグソー・テキストは、複数の先生の授業/研究室や学生の研究で採用されている。このため、フォルダ型の分類やアクセス管理の機能が必要である。Google によるアカウント管理や、Google ドライブのフォルダや共有制御は、十分な操作性と品質であると考えた。

5.2 スプレッドシート API の割り当て

2019 年度前期、情報系の大学 (公立はこだて未来大学) の「情報マネジメント論」授業で [9]、ジグソー・テキストを使ったとき、アプリケーションが重く待たされるという、今回と似た問題が発生した。

原因はスプレッドシートを取得する API (Sheets API) の割り当て (quota、クォータ) であった。データを読むリクエストは、無料枠での利用では、100 秒あたり 1 ユーザあたり 100 回が上限であり、これを増やすには有料で利用する必要がある。ここでユーザとは、この場合はジグソー・テキスト・システム自体に該当する。このままでは、100 秒 (2 分間弱) の間に 100 人の受講者 (ユーザ) しか問題を読み込めないことになってしまう。

5.3 問題データのキャッシュと有効期限

そこで、3.3 節および図 3 で説明した問題データを Google スプレッドシートから取得する処理に、キャッシュを導入して割り当ての制限を回避している。この方が、費用面だけでなく、性能面からも有効と考えた。キャシュには分散型インメモリ・データ・キャッシュの Memcache[10] を使い、キャッシュの有効期限を 100 秒に設定している。6 章で述べるが、システムのスケーリングによって、リクエストを複数のプロセス (インスタンス) で処理することになるが、Memcache のキャッシュはすべてのインスタンスで共有される。

キャッシュの有効期限を長く取れば、問題データ読み込みの性能面では有利である。一方で、有効期限が長いと、Google スプレッドシートへの修正が問題に反映されるまでの時間が長くなる。キャッシュをフラッシュ (削除) するなど強制的に反映する仕組みを実装できるが、問題作成者がその手順を忘れるリスクを伴う。忙しい先生が、演習の直前でも安全に問題を更新できるように、キャッシュの有効期限を割り当て制限回避の最小限にとどめてきた。

5.4 対策: プロセス内キャッシュ

今回、問題データ取得処理が重いことが問題となった。共有メモリ上のキャッシュの有効期限を 100 秒よりも長くするなどの対策も考えられるが、遅くなったのは最初の 2 分 (120 秒) である。この期間に最低 1 回は実行する必要があり、かつ 1〜2 回しか実行されない処理に対する効果には疑問があった。

今回は次の 6 章の検討を踏まえて、この 2 分間に数百回実行される共有メモリ上のキャッシュ処理を軽くするために、プロセス内にも (プロセス間で共有されない) キャッシュを導入した。これは Python プログラムのグローバル変数であり、共有メモリへのアクセスよりも軽いと考えた。

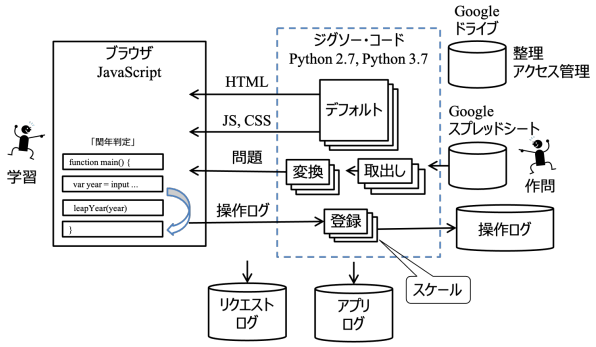

図 13　サービスの集まりで構成するジグソー・コード

6. 課題 2: システム構成の再構築

6.1 クラウド・プラットフォームの性質

Google や Amazon などスケールするクラウド・プラットフォームは、短い処理、数 100 ミリ秒で返す処理の集まりとしてアプリケーションが構成されることを想定して最適化されている。たとえば「App Engine は、1 秒未満でリクエストに応答するレイテンシが短いアプリケーション向けに、自動スケーリングのための容量を予約して」いる [11]。

ジグソー・コードなどは、この点の設計が不十分であった。問題ページの HTML を返す軽い処理も、Google スプレッドシートから問題データを取り出して返す重い処理も、全て同じ 1 つのプロセスが担当していた。

6.2 対策: 複数の小さなサービスに分割

そこで、システムを、ページの HTML を生成する処理、問題データを生成する処理、Google スプレッドシートのデータを取得する処理、操作ログを保存する処理の、4 つのサービスに分割した (**図 13**)。

また、急なアクセス増に対応するため、スケーリングにおけるインスタン数の増減を制御するパラメタを調整した。具体的には、新たなインスタンス (プロセス) を生成するとき、一定数多く生成するようにした。これによって、使っていなくても常にその数のインスタンスが存在することになる。

クラウドに費用はインスタンス数 × 時間で計算されるので、常に一定数のインスタンスが存在すると、月々の使用料金がかさむ。インスタン生成のパラメタは、負荷が大きくなる時間帯を予想して調整したい。

7. 課題 3: 操作ログ送信のリトライ

7.1 検討と開発の経緯

Topic Writer の開発当初、文章本体のデータ保存時に操作ログをまとめて送信していたことがあり、ジグソー・コードでも、この複数の操作をまとめて送信する仕組み使っている。並べ替え操作と送信とが 1 対 1 であるにもかかわらず、送信されたデータの中身を見ないと、プレイの ID や操作の種別が分からない。

7.2 対策: 操作ログ送信のリトライとシリアル番号の付与

操作ログが失われるリスクを小さくするために、送信に失敗したとき、ブラウザ側 (JavaScript) で操作ログの送信をリトライするように修正した。また、リクエスト・ログを見なくても操作ログの欠損を検出できるように、送信にシリアル番号を付与した。リクエスト・ログは、システム管理者でないと見られないが、シリアル番号をデータ分析者やデータ分析アプリケーションが見られるようにした。

7.3 考察: 操作ログの 2 重登録

以前からスケールによって操作ログは複数のプロセスで処理されていたため、DB 登録順序 ≠ 操作順序となり要注意であった。今後はさらに、リトライの導入によって、ブラウザにエラーが返っても実際は登録されてる二重登録の可能性にも、注意が必要となった。

サーバ側で DB を検索して登録済かどうかをチェックしてから登録するようにすれば、2 重登録を防げる。しかし、このようなチェックは処理が重くなるので今回は採用せず、操作ログを分析するアプリケーションや研究者が注意する方式を採用した。

7.4 考察: リアルタイム・モニタリング・システムの性能設計

これに限らず、操作ログを送信・登録するときにデータ加工を伴うリアルタイム・モニタリング・アプリケーションは、性能設計が必要である。例えば、前回操作と今回操作との間で正解との編集距離の差をリアルタイムでモニタするようなシステムは、編集距離の差をいつ・どのプログラムが計算するかが、システムの性能に影響するだろう。

同様に、コンピュータ適応型テスト (Computer-adaptive Testing、CAT) においても、解答パターンに応じて動的に項目を選ぶアルゴリズムや、アルゴリズムをどこで実行するかの性能設計が必要となるだろう。

8. 対策結果の確認

8.1 テスト環境で確認

問題を再現するテスト環境を構築し、対策の効果を確認した。最低限エラーが発生しないこと、またレイテンシ (ユーザ側の応答時間) も改善したことを確認した。

負荷テストには Locust[12] を採用した。ジグソー・コードにかかる負荷を調整した結果、ユーザ (リクエストを送信するプログラム) を 1 秒間に 4 つずつ、200 ユーザまで増やした。各ユーザは、問題ページの HTML、それが読み込む JavaScript と CSS、さらに問題データをリクエストす

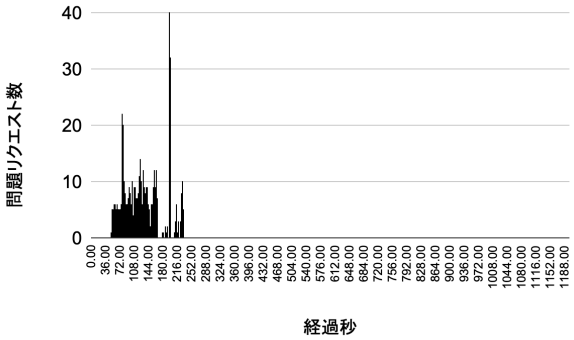

図 14　負荷テスト環境下での問題発生時版へのリクエスト数の推移

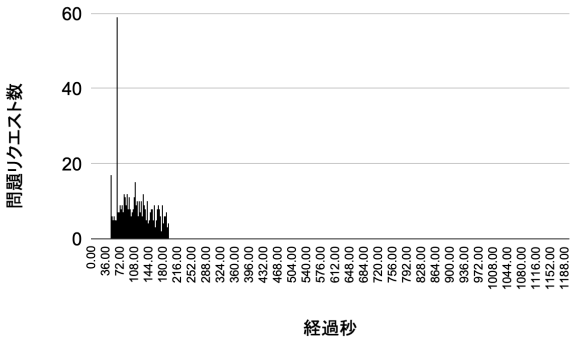

図 15　負荷テスト環境下での対策版へのリクエスト数の推移

る。再読み込みを想定して、これを 4 回繰り返してから、操作ログを 6 回送信した。

まず、負荷が再現されてるかを確認する。**図 14** は問題発生時のジグソー・コードを負荷テスト環境下で実行したときの、リクエスト数の推移である。図 4 と、縦軸の高さと横軸幅を合わせてある。ピークの高さと立ち上がりの急さを再現できている。**図 15** は、同様に対策版のジグソー・コードに対するものだが、縦軸が長く 60 件までである。対策版はレイテンシが小さいので、次のリクエストが早く送信されてしまうためである。すなわち、対策によってスループットが向上したことも分かる。

この負荷下で、問題発生時版では、問題データの取出しで 30 件のエラーが発生した。一方、対策版では 0 件であった。また、ユーザ側の応答時間が、問題発生時版では 5〜30 秒で推移したのに対し、対策版では 2〜9 秒で推移した。2〜9 秒が十分早いかという問題は残るが、対策によって改善が見られた。

8.2　実環境で確認

対策の後、2020 年度後期の授業でジグソー・テキストが使われる機会があり、対策の効果を確認するチャンスとなった。ユーザ数は約 160 人であったが、期待した負荷とはならず、十分な検証とはならなかった。

授業は公立はこだて未来大学の 1 年生向けの「データサイエンス入門」で、データ収集〜分析を体験する題材としてジグソー・テキストをプレイした。受講生は 160 人、オンライ授業であった。授業中に時間を取って、「では、解いてください」と教師が指示を出した。並べ替え結果の正否は成績と無関係であった。問題へのリンクが授業前から受講生に提示されていて、いつでも自由に問題を見たり並べ替え操作することができた。

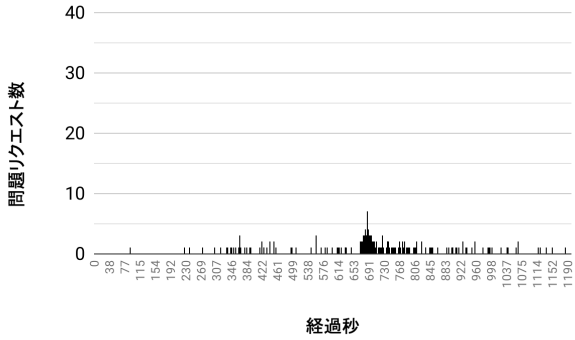

図 16　リクエスト数

図 16 は、問題ページへの秒あたりのリクエスト数で、横軸が経過時間、縦軸がリクエスト数である。図 4 が、約 230 人で 52 秒かけて秒間最大リクエスト数が 33 件/秒に達したのに対し、図 16 は 160 人で 5 分〜10 分かけて最大 7 件/秒に達した。図 4 の方が、秒間最大リクエスト数が人数の差以上に多く、かつ立ち上がり方が急である。この違いは、実施方法の違いによるのではないかと考えられる。対策の効果を確認できるほどの負荷とはならなかった。

9.　考察

今回、Web アプリであるジグソー・コードについて、大学生によるアプリ操作の傾向が明らかになった。特に負荷の高まりでシステムが重くなったときでも、ブラウザの応答を待たずに再読み込みすることが分かった。これは肯定的に捉えるべきであろう。実際、クラウド・プラットフォームが、数 100 ミリ秒で応答するアプリケーションに最適化されてることと一致していると考える。CBT 導入の検討において ICT 機器への慣れが問題にされることがあるが、デジタル・ネイティブ世代の「慣れ」には再読み込みの連打が含まれるかもしれない。

今回はアプリケーション (プロジェクト) を複数のプロセス (サービス) に分割したり、キャシュを多段階で用意することで対策できた。他の対策としては、問題データを動的に生成するのでなく、HTML が読み込む JavaScript や CSS のような静的データとして処理する方式が考えられる。このことは、問題データが作問ユーザによってデータベースに登録されるのでなく、JavaScript プログラムのように、開発者によってデプロイ (ソフトウェアのアップデート) されることを意味する。問題データはソフトウェアの一部となり、運用上の柔軟性や効率を犠牲にすること

となる。

あるいは、問題データを端末に事前に配信することで、本稿のような試験開始時の負荷を回避できる。ネイティブ・アプリケーションであれば、そのような実装は容易であろう。

クラウドの費用がインスタンス数 × 時間で計算されると指摘した。CBT 導入の検討では、DDoS 攻撃によるテスト妨害の排除が課題としてあげられている [7] が、サーバ運用費用の増加という害もある。

10. おわりに

10.1 まとめ

今回は、大学生による Web アプリ操作の傾向の一端を、定量的に明らかにできた。特に負荷の高まりでシステムが重くなったときでも、ユーザはブラウザの応答を待たずに再読み込みすることが分かった。これは、クラウド・プラットフォームが、数 100 ミリ秒で応答するアプリケーションに最適化されてることと一致していると考える。このようなシステムでは、操作ログに限らず、ユーザ操作による再読み込みを含めて、リトライを前提にシステムを設計する必要があるだろう。

ユーザ数の評価も注意が必要であることを、具体的に示すことができた。1 人あたり 7 回問題を読み込むのであれば、7 倍のユーザが 1 回ずつ読み込むのと変わらない。ユーザ操作を測定・分析する学習分析システムは、数 10 分といった時間スパンでのユーザ数だけでなく、秒単位のアクセス数やアクセス数増加の勾配を見積もって設計する必要がある。すなわち「前に 300 人で使って大丈夫だったから、今回は 200 人なので大丈夫」と単純には言えない。

本件では、テスト環境で負荷対策の効果を確認できた。負荷の定量的な情報があるからこそ、このように確認できるのである。

10.2 残された課題

本システムでは、リトライの導入によって操作ログが重複して登録される可能性が生じたので、今後は分析にあたって考慮する必要がある。

本研究に限らず、学習分析システム全般で、学生のアプリ操作やアクセス状況全般について、システム設計の観点からも、定量的な知見のさらなる積み重ねが必要であろう。

謝辞 青山学院大学社会情報学部の藤井沙苗氏には、ジグソー・コードのパズル問題を提供していただいた。本研究は JSPS 科研費 20H01728 の助成を受けたものである.

参考文献

[1] 森永 笑子 , 松本 慎平 , 村上 瑠香 , 林 雄介 , 平嶋 宗, カード操作方式によるプログラミング学習支援システムでの学習過程の可視化方法の提案, 人工知能学会 先進的学習科学と工学研究会 85, 92-97, 2019-03-07
http://id.nii.ac.jp/1004/00009654/

[2] 中村 陽太, 大場 みち子, 山口 琢, 伊藤 恵, 学習進度に対応するパズルを利用したプログラミング思考過程の分析, 情報処理学会研究報告コンピュータと教育 (CE), 2019-CE-151(1), 2019-09-28
http://id.nii.ac.jp/1001/00199559/

[3] 森永 笑子, 松本 慎平, 林 雄介, 平嶋 宗, カード操作方式によるプログラミング学習支援システムにおける学習者の活動に基づく学習課題の特徴分析, 電気学会 C 部門 情報システム研究会 (CIS), 2019-11-10
http://id.nii.ac.jp/1031/00126864/

[4] 中村 陽太, 大場 みち子, 山口 琢, 伊藤 恵, 授業進度に対応するパズルを利用したプログラミング思考過程の分析と教育支援システムの開発, 情報処理学会 第 82 回全国大会講演論文集, 2020-02-20
http://id.nii.ac.jp/1001/00205909/

[5] Salil Maharjan and Amruth Kumar, Using Edit Distance Trails to Analyze Path Solutions of Parsons Puzzles, Educational Data Mining 2020 (EDM 2020), 2020-07-10
https://educationaldatamining.org/files/conferences/EDM2020/papers/paper_163.pdf

[6] 伊東大輝, 島川博光, コードパズルによるプログラミング的思考力を考慮した理解度の推定, 情報科学技術フォーラム (FIT2020) 講演論文集, 2020

[7] 平成 28～30 年度文部科学省 大学入学者選抜改革推進委託事業 情報学的アプローチによる「情報科」大学入学者選抜における評価手法の研究開発最終成果報告書
別添資料 6 大規模 CBT システム構築への課題とその解決策
https://www.mext.go.jp/content/1412881_5_1.pdf

[8] 林浩一, 山口琢, 大場みち子, ロジカルシンキングにおける目的と手段が反転する誤答発生の過程分析, 情報処理学会 研究報告コンピュータと教育 (CE), 2019-CE-152(21), 2019-11-08
http://id.nii.ac.jp/1001/00200286/

[9] 大場みち子, 山口琢, 情報科目における作文行動の記録ツールを適用した反転授業の教育実践, 情報処理学会 研究報告コンピュータと教育 (CE), 2020-CE-154, 2020-03-07
http://id.nii.ac.jp/1001/00204103/

[10] Google, Memcache の概要
https://cloud.google.com/appengine/docs/standard/python/memcache?hl=ja

[11] Google App Engine, リクエストの処理方法
https://cloud.google.com/appengine/docs/standard/python/how-requests-are-handled

[12] Locust
https://locust.io/

[13] 山口琢, 大場みち子, 高橋修, 相互運用可能な作文計測システムの設計, 電気学会 【C】電子・情報・システム部門 情報システム研究会, IS14033, 2014-09-25
http://id.nii.ac.jp/1031/00077439/