function [calls, paths, ss] = ggisr_retry_sim_ac(sysinfo, sysstate, calls, callmax)
% function [calls, paths, ss] = ggisr_retry_sim_ac(sysinfo, sysstate, calls, callmax)
% A simulation for a G(t)/GI(t)/s/r + retrials queue.
% No Revisits!
% This version uses an array (hence the _a),
% instead of a heap, for event management.
% It also relies on outside random number generators for all randomness
% for each _c_all, so we can use _c_ommon random numbers.
% sysinfo contains:
% int nservers
% int nbuf: number of buffer spaces (not incl. servers)
% double maxtraffic: (in Erlangs), to hint how big orbit might get
% No Longer Used:
% function_handle a_func: inter-arrival function
% function_handle svc_dur_func: takes arguments nrows, ncols, distrib_struct
% function_handle r_dur_func: takes arguments nrows, ncols, distrib_struct, type
% double[2] v_probs: revisit probs, v_probs(1,0+1)=Pr{revisit after 1st svc, type 0}
% calls contains:
% int ncalls: usually =length(arriv)=length(svc_dur) etc,
%  but sometimes those arrays will be longer than ncalls, and have unused parts.
% int n_outside_calls: number of calls originating from outside,
%  that is, revisits don't count.
% double[] arriv: arrival epoch for each call
% double[] svc_dur: service duration for each call
% double[] ret_durs; if any are <0 that indicates quit-retrying.
% int[] b_vals: balking threshold: balk if >= that number in queue.
%  e.g. if =0, will always balk
% boolean[] timeout: did it timeout?
% int[] n_tries
% int[] n_visits
% double[] qtime: the epoch that this call entered the queue
% double[] wq: duration of wait in the queue itself
% double[] wo: duration of wait in orbit
% no longer used:int[] parent (often sparse): ID number of the parent of this call
% no longer used:int[] child (often sparse): ID number of the child of this call
% Okay, we need clear definitions for n_visits and n_tries:
% A try is any attempt to enter service or the queue.
% Thus, every call tries at least once (even if it balks & doesn't enter orbit).
% A visit is a successful entry into the queue or straight into service.
% Thus, a call that tries but is blocked or balks and doesn't come back,
% or enters orbit, retries, and gives up, has n_visits = 0.
% A revisiting call inherits its parent's n_visits,
% but not n_tries.
% n_visits is only updated when someone enters the queue
% or enters service directly (w/o waiting in the queue),
% n_tries is only updated when someone tries or retries, _not_ upon
% entry into orbit.
% paths contains
% int path_size (number of things in the path)
% double[] epochs
% char[] etypes
% int[] n_orbit
% int[] n_sq (service plus queue)
% the n_ columns are the value just _after_ the associated epoch
% Event types (internal):
% a = arrival from outside
% s = service completion, no revisit
% r = retrial from orbit
% e = error, should not be seen!
% x = end-of-simulation
% Event result types
% A = arrival from outside enters sq
% a = arrival from outside balks, enters orbit
% b = arrival from outside balks, gives up
% S = service complete, no revisit
% v = service complete, revisit -> orbit
% R = retry from orbit, enters sq
% o = retry from orbit, balks, enters orbit
% r = retry from orbit, balks, gives up

% ss is the output version of sysstate. Both contain:
% initialize_me: 0 if things are already initialized, 1 if ggisr_retry_sim
% should initialize for itself.
% int n_sq : number in servers+queue
% int n_o : number in orbit
% double CLK : the system clock
% int[] callq : array of which calls are in the queue, length=nbuff
% int qhead : integer saying which call in the queue is at the front;
%   eg qhead=10, then callq(10+1)=ID of first call, callq(11+1) = ID of 2nd call
% int qlast : integer saying which call is last in the queue, callq(qlast+1)
% the +1) is so qhead, qlast can be zero-based, easier for mod()
% If there's just one call in the queue, qlast==qhead.

% double[3][] etimes
% etimes is a 3-by-many array of event times. Default values should be +Inf.
% row 1 is for arrivals (we'll only really use the first element of that row),
% row 2 is for service completions
% row 3 is for retrials
% int[3][] ecalls
% is an array of the same size that holds call ID's for the corresponding
% event times. Note that we don't have to store event types--they are implicit
% in which row of the array something is stored at.
etypelist='asr'; % arrivals, services, retrials

si = sysinfo;
ss = sysstate;
calls.ncalls = 0;
calls.n_outside_calls = 0;
% preallocate memory for these, so we don't spend time increasing their
% allocations during the while-loop.
guess_ncalls = length(calls.arriv);

calls$arriv = NaN * ones(guess_ncalls,1);
calls$servDur = NaN * ones(guess_ncalls,1);
calls$timeout = NaN * ones(guess_ncalls,1);
calls$n_tries = NaN * ones(guess_ncalls,1);
calls$qtime = NaN * ones(guess_ncalls,1);
calls$wq = NaN * ones(guess_ncalls,1);
calls$wo = NaN * ones(guess_ncalls,1);
calls$n_visits = NaN * ones(guess_ncalls,1);
calls$parent = sparse(guess_ncalls,1);
calls$child = sparse(guess_ncalls,1);

% initialize the paths variable
paths.path_size = 0;
guess_num_events = callmax * 4;
paths.etypes(guess_num_events) = 'e';
paths.epochs = zeros( guess_num_events, 1);
paths.n_orbit = zeros( guess_num_events, 1);
paths.n_sq = zeros( guess_num_events, 1);
paths.guess_num_events = guess_num_events;
pathlen = length(paths.epochs);
nservers = si.nservers;
ncols_ret_durs = size(calls.ret_durs, 2);

% If we're told to initialize for ourselves, schedule that first arrival.
if( ss.initialize_me == 1 )
    ss.qhead = 0;
    ss.qlast = -1;
    % [iat] = feval(si.a_func, 1, 1, si.a_dist, ss.CLK); % iat = inter-arrival time
    tmp = ss.CLK + iat;
    n = min(find(calls.arriv >= ss.CLK) ); % find first call to arrive
    % after initial CLK value.
    tmp = calls.arriv(n);
    if( n == callmax )
        flag = 1;
    end
end
calls.ncalls = n;
calls.n_outside_calls = 1;
calls.n_tries(n) = 1;
calls.n_visits(n) = 0;
calls.wo(n) = 0;
calls.wq(n) = 0;
% get service duration, call it "sdur"
[dur, timeout] = feval(si.svc_dur_func, 1, 1, si.svc_dur_dist);
sdur = calls.svc_dur(n);

% guess a heap size
rho = si.maxtraffic / si.nservers;
if( max(size(calls.ret_durs))>1 && all(calls.ret_durs(:,1) == 0) )
    guess_max_orbit_size = 0;
elseif( rho < 1 )
    guess_max_orbit_size = 2 * rho / (1-rho) ;
else % rho>= 1
    guess_max_orbit_size = 5*(si.maxtraffic - si.nservers);
end
guess_max_heap_size = si.nservers + si.nbuf + guess_max_orbit_size + 1;
guess_max_heap_size = ceil(guess_max_heap_size);
% set up the array that keeps track of event times
etimes = inf * ones(3, round(max(guess_max_orbit_size, si.nservers)));
% set up the array that keeps track of event CallID
calls = NaN * ones(size(etimes));

% and schedule the first event, an arrival
etimes(1,1) = tmp;
ecalls(1,1) = n;
end

old_CLK = 0;
% get next event
[tmp_times, indcs] = min(etimes, [], 2); % the 2 means operate on rows
[ss.CLK, index] = min(tmp_times);
etype = etypelist(index);
ecall = round(ecalls(index, indcs(index)));
flag = 0;
while( not( flag ) )
    % etimes
    % ecalls
    fprintf(1, 'CLK %g type %s ecall %d\n', ss.CLK, etype, ecall);
    fprintf(1, 'n sq %d n o %d\n', ss.n_sq, ss.n_o);
    fprintf(1, 'CLK - old_CLK %g\n', ss.CLK - old_CLK);
%etype

switch etype
    case 'x' % the ending event
        etype_report = 'x';
        flag = 1;
    end

    case 'a' % arrival from outside
        etype_report = 'e'; % just for now
        % create next call
        [iat] = feval(si.a_func,1,1,si.a_dist,ss.CLK); % iat = inter-arrival time
        tmp = ss.CLK + iat;
        n = calls.ncalls + 1;
        tmp = calls.arriv(n);
        calls.n_outside_calls = calls.n_outside_calls + 1;
        if( n == callmax)
            flag = 1;
        end
        calls.ncalls = n;
        calls.n_tries(n) = 1;
        calls.n_visits(n) = 0;
        calls.wq(n) = 0;
        calls.wq(n) = 0;
        % get service duration, call it "sdur"
        [dur, timeout] = feval(si.svc_dur_func,1,1,si.svc_dur_dist);
        calls.svc_dur(n) = dur;
        sdur = calls.svc_dur(n);
        % schedule next arrival
        etimes(1,1) = tmp;
        ecalls(1,1) = n;
        % now deal with the call that just arrived:
        % four possibilities: enter svc, enter q,
        % enter orbit (balk & retry), or give up(balk&not retry)
        enter_svc = 0;
        % just to initialize it
        if( ss.n_sq < nservers ) % enter svc
            enter_svc = 1;
        else
            % servers are full, might balk
            retry = 0; % just to initialize it
            if( ss.n_sq == nservers+si.nbuf)
                balk = 1;
            else
                balk = ss.n_sq - nservers >= calls.b_vals(ecall);
            end
        end
        if( balk )
            % calls.n_tries(ecall) should be 1
            % since this is a fresh outside arrival
            % but we'll do it this way anyway.
            %retry = calls.ret_durs(ecall,calls.n_tries(ecall)) >= 0;
            % actually, we'll shortcut it to make things go faster.
            ret_dur = calls.ret_durs(ecall,1);
            retry = ret_dur >= 0;
        end
    end
    if( enter_svc )
        etype_report = 'A'
        ss.n_sq = ss.n_sq + 1;
        % add a svc event
        sdur = calls.svc_dur(ecall);
        % find an open server
        f = find(isinf(etimes(2,:)));
        f = f(1); % just the first open server
        etimes(2,f) = ss.CLK + sdur;
        ecalls(2,f) = ecall;
% calls.n_visits(ecall) = calls.n_visits(ecall)+1;
% Since no revisits are allowed, n_visits = 0 or 1
% So we don't need to increment it, just set it to 1.
calls.n_visits(ecall) = 1;
elseif( not(balk) ) \% and not enter_svc: must
% enter the queue
etype_report = 'A';
ss.n_sq = ss.n_sq + 1;
ss.qlast = mod(ss.qlast+1,si.nbuf);
ss.callq(ss.qlast+1) = ecall;
% don't schedule a new event
calls.qtime(ecall) = ssCLK;
% Since no revisits are allowed, n_visits = 0 or 1
% So we don't need to increment it, just set it to 1.
calls.n_visits(ecall) = 1;
elseif( balk \&\& retry ) \% enter orbit
etype_report = 'a';
ss.n_o = ss.n_o + 1;
% add a retry event
%[dur] = feval(si.r_dur_func,1,1,si.r_dur_dist);
%dur = ret_dur;

%%%%%%%%%%%%%%%%%%%%%%%%%
% find an open retrial event
f = find(isinf(etimes(3,:)));% the tricky part is that the default
if( length(f) > 0 ) \% values need to be Inf rather than 0.
    f = f(1); \% just the first open retrial
else \% need to make the event calendar bigger
    % Also, want to expand by more than just 1 slot,
    % so we don't spend too much time expanding.
    oldlen = length(etimes(3,:));
    addlen = round(0.5*oldlen);
    etimes = [etimes, Inf*ones(3,addlen)];
    ecalls = [ecalls, NaN*ones(3,addlen)];
    f = 1+oldlen;
end
etimes(3,f) = ssCLK + ret_dur;
ecalls(3,f) = ecall;

%%%%%%%%%%%%%%%%%%%%%%%%%
% this new arrival didn't get served--don't update their # of visits
% wait until they retry to update their n_tries
% calls.n_visits(ecall) = calls.n_visits(ecall)+1;
calls.n_tries(ecall) = calls.n_tries(ecall)+1;
calls.wo(ecall) = ret_dur;
else \% balk and not retry = give up
    \% no new event to schedule
    \% update any call data?
    etype_report = 'b';
end
%%%%%%%%%%%%%%%%%%%%%%%%%'

\textbf{case 's'} \% service completion
etype_report = 'S';
% erase the service completion event
etimes(2,indcs(indx)) = Inf;
ecalls(2,indcs(indx)) = NaN;
% might re-visit; prob depends on if it timed out
%disp('calls.n_visits(ecall)')
calls.n_visits(ecall)
revisit_prob = si.v_probs( calls.n_visits(ecall), ... 
1+calls.timeout(ecall));
revisit = rand(1) < revisit_prob;

% No revisits!
%
if( revisit )
  etype_report = 'v';
  % generate a new call record
  par = ecall; % parent
  chi = calls.ncalls + 1; % child
  calls.ncalls = chi;
  calls.n_tries(chi) = 1;
  calls.n_tries(chi) = 0;
  % only as many visits as the parent call,
  % which might get updated if chi gets into svc.
  calls.n_visits(chi) = calls.n_visits(par);
  calls.wq(chi) = 0;
  calls.parent(chi) = par;
  calls.child(par) = chi;
  % get service duration, call it "dur"
  [sdur, timeout] = feval(si.svc_dur_func,1,1,si.svc_dur_dist);
  calls.svc_dur(chi) = sdur;
  calls.timeout(chi) = timeout;
  calls.arriv(chi) = tmp;
  calls.arriv(chi) = ss.CLK+sdur;
  % schedule retry event
  [ret_dur] = feval(si.r_dur_func,1,1,si.r_dur_dist);
  ret_dur = 1/0; % cause a warning
end % if revisit

%%%%%%%%%%%%%%%%%%%%%
% find an open retrial event
f = find(isinf(etimes(3,:)));
if( length(f) > 0 )
  f = f(1); % just the first open retrial
else % need to make the event calendar bigger
  % the tricky part is that the default
  % values need to be Inf rather than 0.
  % Also, want to expand by more than just 1 slot,
  % so we don't spend too much time expanding.
  oldlen = length(etimes(3,:));
  addlen = round(0.5*oldlen);
  etimes = [etimes, Inf*ones(3,addlen)];
  ecalls = [ecalls, NaN*ones(3,addlen)];
  f = 1+oldlen;
end
etimes(3,f) = ss.CLK + ret_dur;
ecalls(3,f) = chi;
ss.n_o = ss.n_o + 1;

%%%%%%%%%%%%%%%%%%%%%
% should we really update the parent's number of tries here? no!
% calls.n_tries(par) = calls.n_tries(par)+1;
% and add to the wait time in orbit
% calls.wo(chi) = ret_dur;
end % if revisit
%
% and now pull someone new into service
if( ss.n_sq > nservers )
  n = ss.callq(ss.qhead+1);
  calls.wq(n) = ss.CLK - calls.qtime(n);
  % increment the pointer to the head of the queue
  ss.qhead = mod(ss.qhead+1,si.nbuf);
  % schedule the service completion event
  sdur = calls.svcDur(n);
  % don't need to find an open server;
  % we know where the just-finished service was.
  etimes(2,inds( indx )) = ss.CLK + sdur;
ecalls(2, indcs(indx)) = n;
end
% Finally, decrement the number in service+queue
ss.n_sq = ss.n_sq - 1;

% retrial from orbit
case 'r'
% now deal with the call that just arrived:
% four possibilities: enter svc, enter q,
% enter orbit (balk & retry), or give up(balk&not retry)

% need to update n_tries; that was not done
% when it entered orbit.
calls.n_tries(ecall) = calls.n_tries(ecall) + 1;

% now determine what happens for this retrial.
enter_svc = 0; % just to initialize it
if( ss.n_sq < nservers ) % enter svc
  enter_svc = 1;
else
  retry = 0; % just to initialize it
  if( ss.n_sq == nservers+si.nbuf)
    balk=1;
  else
    balk= ss.n_sq - nservers >= calls.b_vals(ecall);
  end
end
% erase the retrial event
etimes(3, indcs(indx)) = Inf;
ecalls(3, indcs(indx)) = NaN;
% and now process the event
if( enter_svc )
etype_report = 'R';
ss.n_sq = ss.n_sq + 1;
ss.n_o  = ss.n_o - 1;
% add a svc event
sdur = calls.svc_dur(ecall);
% find an open server
f = find(isinf(etimes(2,:)));
f = f(1); % just the first open server
etimes(2,f) = ss.CLK + sdur;
ecalls(2,f) = ecall;
calls.n_visits(ecall) = calls.n_visits(ecall)+1;
% Since no revisits are allowed, n_visits = 0 or 1
% So we don't need to increment it, just set it to 1.
calls.n_visits(ecall) = 1;
elseif( not(balk) )
% enter the queue
etype_report = 'R';
ss.n_sq = ss.n_sq + 1;
ss.n_o  = ss.n_o - 1;
ss.qlast = mod(ss.qlast+1,si.nbuf);
ss.callq(ss.qlast+1) = ecall;
% don't schedule a new event
calls.qtime(ecall) = ss.CLK;
% Since no revisits are allowed, n_visits = 0 or 1
% So we don't need to increment it, just set it to 1.
calls.n_visits(ecall) = 1;
elseif ( balk && retry )
% re-enter orbit
etype_report = 'o';
% re-establish that retry event
% (nice that we know one is free, instead
% of possibly expanding the list)
%[dur] = feval(si.r_dur_func,1,1,si.r_dur_dist);
% already generated retry dur
etimes(3,indcs(indx)) = ss.CLK + ret_dur;
ecalls(3,indcs(indx)) = ecall;
calls.wo(ecall) = calls.wo(ecall)+ret_dur;
else % balk and not retry = give up
etype_report = 'c';
ss.n_o = ss.n_o - 1;
% no new event to schedule
end
end
paths.path_size = paths.path_size + 1;
if ( n > pathlen )
 addlen = round(0.5 * n);
% more events in the sample path than we expected
paths.epochs = [paths.epochs ; NaN*ones(addlen,1)];
paths.etypes(n+addlen) = 'e';
paths.n_orbit = [paths.n_orbit; NaN*ones(addlen,1)];
paths.n_sq = [paths.n_sq ; NaN*ones(addlen,1)];
pathlen = length(paths.epochs);
end
paths.epochs(n) = ss.CLK;
paths.etypes(n) = etype;
paths.n_orbit(n) = ss.n_o;
paths.n_sq(n) = ss.n_sq;
%the n_ columns are the value just _after_ the associated epoch
old_CLK = ss.CLK;
% get next event
[tmp_times, indcs] = min(etimes,[],2); % the 2 means operate on rows
[ss.CLK, indx] = min(tmp_times);
etype = etypelist(indx);
ecall = round(ecalls(indx,indcs(indx)));
% the round() should be redundant,
% but hopefully it will convince matlab that it's an integer, and okay to use
% as an array index.
end % main while loop

% wrap-up: take remaining events off the heap, and
% note them as "leftover"
for ni = 1:size(etimes,1) % for each row of the etimes list
 mine = not(isinf(etimes(ni,:)));
 if( any(mine) )
  callids = round(ecalls(ni,mine));
  calls.n_tries(callids) = NaN;
  calls.n_visits(callids) = NaN;
  calls.wq(callids) = NaN;
  calls.wo(callids) = NaN;
 end
end
% Also, truncate the arrays in the "path" structure.
n=paths.path_size;
paths.epochs = paths.epochs(1:n);
paths.etypes = paths.etypes(1:n);
paths.n_sq = paths.n_sq(1:n);
paths.n_orbit = paths.n_orbit(1:n);

% And unused calls in the "calls" structure
n=calls.ncalls;
calls.arriv = calls.arriv(1:n);
calls.svc_dur = calls.svc_dur(1:n);
calls.timeout = calls.timeout(1:n);
calls.n_tries = calls.n_tries(1:n);
calls.qtime = calls.qtime(1:n);
calls.wq = calls.wq(1:n);
calls.wo = calls.wo(1:n);
calls.n_visits = calls.n_visits(1:n);
calls.parent = calls.parent(1:n);
calls.child = calls.child(1:n);